

Signature Embedding

A Deep Metric Learning Approach to
Writer Independent Off-Line Signature
Verification

Einbetten von Unterschriften

*Schreiberunabhängige Verifizierung statischer
Unterschriften mithilfe tiefer neuronaler Netze*

by

Hannes Rantzsch

A thesis submitted to the
Hasso Plattner Institute
at the University of Potsdam, Germany
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE IN IT-SYSTEMS ENGINEERING

Supervisors

Prof. Dr. Christoph Meinel
Dr. Haojin Yang

Internet-Technologies and Systems
Hasso Plattner Institute
University of Potsdam, Germany

August 18, 2016

Abstract

The handwritten signature is widely employed and accepted as a proof of a person's identity. In our everyday life, it is often verified manually, yet only casually. As a result, the need for automatic signature verification arises.

In this thesis, we propose and explore a new approach to the writer independent verification of off-line signatures. Our approach, named *Signature Embedding*, is based on deep metric learning. Comparing triplets of two genuine and one forged signature, our system learns to embed signatures into a high-dimensional space, in which the Euclidean distance functions as a metric of their similarity.

We investigate how such a distance metric can be employed to verify the genuineness of signatures. We explore applicable evaluation metrics and evaluate the performance of our system.

We compare the evaluation results to the results of the ICDAR SigWiComp 2013 challenge on off-line signature verification. Our system achieves a top-ranking performance in both of the tasks in the challenge, outperforming the systems that participated in the challenge in nearly all respects.

Zusammenfassung

Handgeschriebene Unterschriften sind ein verbreitetes Mittel, um die Identität einer Person zu belegen. In unserem Alltag werden sie oft manuell, jedoch nur flüchtig geprüft. Aus diesem Umstand ergibt sich der Bedarf nach einem System zur automatischen Prüfung von Unterschriften.

In dieser Masterarbeit zeigen wir einen neuen Ansatz zur Verifizierung von statischen Unterschriften, der unabhängig vom Verfasser der Unterschrift eingesetzt werden kann. Der Ansatz basiert auf dem tiefen Lernen von Metriken („Deep Metric Learning“). Wir nennen unser System *Signature Embedding* – Einbetten von Unterschriften. Indem es Dreiergruppen von zwei echten und einer gefälschten Unterschrift vergleicht, lernt es die Unterschriften in einen hochdimensionalen Raum einzubetten. In diesem Raum kann der euklidische Abstand der Unterschriften als ein Maß ihrer Ähnlichkeit verwendet werden.

Wir untersuchen, wie ein solches Abstandsmaß genutzt werden kann, um die Echtheit von Unterschriften zu prüfen. Des Weiteren ermitteln wir geeignete Evaluationsmetriken, mithilfe derer die Güte unseres Systems evaluiert werden kann.

Wir vergleichen unsere Evaluationsergebnisse mit den Ergebnissen des „ICDAR SigWiComp 2013“-Wettbewerbs für die Verifizierung von statischen Unterschriften. Es zeigt sich, dass unser System herausragende Ergebnisse in beiden Aufgaben des Wettbewerbs erzielt und die am Wettbewerb teilnehmenden Systeme in nahezu allen Belangen übertrifft.

Acknowledgments

First, I would like to express my gratitude to my supervisors Prof. Dr. Christoph Meinel and Dr. Haojin Yang. I want to thank them for giving me the opportunity to research this interesting topic, as well as for their guidance and advice. I owe many inspiring discussions on the topics of signature verification and deep learning to Dr. Yang. He often helped me find the right direction to proceed with my work.

I also want to thank my dear friends and colleagues Christian Bartz, Dimitri Korsch, and Johannes Jasper. They helped me to refine my understanding of the topic, with the toughest (and sometimes simplest) programming issues, and to come up with new ideas and perspectives on the problem. Last but not least, I owe to them a great time in the office during my work on this thesis.

Finally, I want to thank my family and my dearest Trang in particular for supporting and encouraging me at all times.

Thank you.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Signatures as a Means of Biometric Authentication | 1 |
| 1.1.1 | Automatic Signature Verification | 2 |
| 1.1.2 | Ethical Considerations | 3 |
| 1.2 | Contributions | 4 |
| 1.3 | Outline of the Thesis | 5 |
| 2 | The Signature Verification Problem | 7 |
| 2.1 | Signature Verification | 7 |
| 2.1.1 | Types of Forgeries | 7 |
| 2.1.2 | On-Line and Off-Line Signature Verification | 8 |
| 2.1.3 | Off-Line Signature Verification | 8 |
| 2.2 | Task Specification in This Thesis | 10 |
| 3 | Theoretical Background | 11 |
| 3.1 | Machine Learning | 11 |
| 3.1.1 | Types of Machine Learning | 11 |
| 3.1.2 | Supervised Learning: Linear Regression | 12 |
| 3.1.3 | Classification | 13 |
| 3.1.4 | The Loss Function | 14 |
| 3.1.5 | Adjusting the Parameters | 14 |
| 3.2 | Artificial Neural Networks | 16 |
| 3.2.1 | The Perceptron | 16 |
| 3.2.2 | The Multilayer Perceptron | 17 |
| 3.2.3 | Adjusting the Parameters | 18 |
| 3.3 | Deep Neural Networks | 18 |
| 3.3.1 | Building Blocks of Deep Neural Networks | 19 |
| 3.3.2 | DNNs and GPUs | 21 |
| 3.4 | Similarity Metrics | 22 |
| 3.4.1 | Similarity Metrics and Machine Learning | 22 |
| 3.4.2 | Improvements of Similarity Metrics | 23 |
| 4 | Datasets, Software and Hardware Resources | 25 |
| 4.1 | Datasets | 25 |
| 4.1.1 | MCYT100 | 25 |
| 4.1.2 | GPDSsyntheticSignature | 26 |
| 4.1.3 | SigWiComp2013 | 26 |
| 4.1.4 | MNIST | 26 |

| | | |
|----------|---|-----------|
| 4.2 | Software | 27 |
| 4.3 | Hardware | 27 |
| 5 | Writer Independent Signature Verification: Signature Embedding | 29 |
| 5.1 | Approach and Intuition | 29 |
| 5.2 | Embedding the Signatures | 30 |
| 5.3 | Calculating the Distance | 31 |
| 5.4 | Making a Decision | 32 |
| 5.4.1 | Hard Decisions: Thresholding | 32 |
| 5.4.2 | Soft Decisions: Likelihood Ratios | 33 |
| 5.4.3 | Application Independent Systems | 34 |
| 6 | Implementation | 35 |
| 6.1 | Neural Network Design | 35 |
| 6.2 | Augmentation and Preprocessing | 37 |
| 6.3 | Pretraining | 37 |
| 6.3.1 | Writer Dependent Signature Verification | 38 |
| 6.3.2 | Writer Dependent Training | 38 |
| 6.3.3 | Using the Pretrained Model | 38 |
| 6.4 | Training the Model | 39 |
| 6.4.1 | Batch Sampling | 39 |
| 6.4.2 | Forward Pass and Parameter Update | 40 |
| 6.4.3 | Validation | 43 |
| 6.4.4 | Epochs | 44 |
| 7 | Evaluation | 45 |
| 7.1 | Setup | 45 |
| 7.1.1 | Data | 45 |
| 7.1.2 | Training of the Neural Network Model | 45 |
| 7.1.3 | Validation of the Training Process | 47 |
| 7.1.4 | Number of Reference Signatures | 48 |
| 7.2 | Evaluation Metrics | 48 |
| 7.2.1 | Application Dependent Evaluation | 48 |
| 7.2.2 | Application Independent Evaluation | 50 |
| 7.2.3 | Visualization | 51 |
| 7.3 | Results | 51 |
| 7.3.1 | PCA Visualization | 51 |
| 7.3.2 | Distribution of Distances | 53 |
| 7.3.3 | Application Dependent Evaluation Results | 53 |
| 7.3.4 | Application Independent Evaluation Results | 56 |
| 7.4 | Discussion and Comparison | 56 |
| 7.5 | Experiments | 58 |
| 7.5.1 | Augmentation with a Background | 58 |
| 7.5.2 | Ratio of Hard Triplets | 59 |
| 7.5.3 | Number of Reference Signatures | 61 |

| | |
|--------------------------------------|-----------|
| 8 Related Work | 63 |
| 8.1 Handcrafted Features | 63 |
| 8.2 Deep Learning | 65 |
| 9 Conclusion | 67 |
| 9.1 Learnings and Insights | 68 |
| 9.2 Future Work | 68 |

List of Figures

| | |
|---|----|
| 2.1 Superimposed genuine signatures of one user | 9 |
| 3.1 Underfitting and overfitting in polynomial regression | 13 |
| 3.2 The perceptron | 17 |
| 3.3 The multilayer perceptron | 18 |
| 3.4 The convolution | 20 |
| 5.1 Embedding of signatures into Euclidean space | 30 |
| 5.2 Embedded signatures and forgeries form clusters | 32 |
| 5.3 Trade-off between false accepts and false rejects depending on the threshold | 33 |
| 6.1 Calculation of triplet loss | 41 |
| 6.2 Effect of training the network model on the embedding of triplets . . | 42 |
| 7.1 Validation of the training process | 47 |
| 7.2 PCA visualizations | 52 |
| 7.3 Histograms of distances from the Dutch evaluation dataset | 54 |
| 7.4 Histograms of distances from the Japanese evaluation dataset | 54 |
| 7.5 ROC curve for the Dutch evaluation dataset | 55 |
| 7.6 ROC curve for the Japanese evaluation dataset | 55 |
| 7.7 Signature samples augmented by background structure | 58 |
| 7.8 Comparison of validation metrics for different hard triplet ratios . . . | 59 |
| 7.9 PCA of embedded signatures, trained with hard triplets only | 60 |

List of Tables

| | |
|--|----|
| 4.1 Overview of the datasets we used | 25 |
| 6.1 Layout of the deep neural network | 36 |
| 7.1 Signatures from the ICDAR SigWiComp2013 datasets | 46 |
| 7.2 Application dependent evaluation results | 57 |
| 7.3 Application independent evaluation results | 57 |
| 7.4 Comparison to ICDAR SigWiComp2013 Task 1 | 57 |
| 7.5 Comparison to ICDAR SigWiComp2013 Task 2 | 57 |
| 7.6 Comparison of evaluation metrics for different hard triplet ratios . . . | 60 |
| 7.7 Impact of the number of reference signatures | 61 |

List of Abbreviations

| | |
|------|--|
| ANN | artificial neural network |
| BFS | boosting feature selection |
| DNN | deep neural network |
| EER | equal error rate |
| FAR | false accept rate |
| FC | fully connected |
| FRR | false reject rate |
| GPU | graphics processing unit |
| HOG | histogram of oriented gradients |
| LBP | local binary patterns |
| MLP | multilayer perceptron |
| MSE | mean squared error |
| NIST | National Institute of Standards and Technology |
| PCA | principal component analysis |
| PDF | probability density function |
| ReLU | Rectified Linear Unit |
| ROC | receiver operating characteristic |
| SGD | stochastic gradient descent |
| SVM | support vector machine |
| TAR | true accept rate |

1 Introduction

The handwritten signature is a widely employed method to verify a person's identity in our everyday life. It plays an important role in the legitimation of legal contracts, is used to authorize transactions of money, and serves as an evidence to the provenance of documents. As a part of these processes, a large number of signatures is verified daily, often by visual, human inspection. This verification is done only casually in most cases—especially in everyday scenarios, such as at the supermarket checkout counter—and the signature's correctness is not questioned until legal issues arise.

This situation motivates the creation of automatic signature verification systems. Such systems are required to be both robust and accurate due to the widespread and momentous employment of handwritten signatures in our society. In consequence, automatic signature verification systems have been broadly researched [58].

Nonetheless, signature verification in a realistic scenario remains an interesting problem with many research questions unanswered. In such a scenario, the signature is written with an ordinary pen on a regular sheet of paper, with no additional appliances that capture additional information about the signing process. Furthermore, signatures of arbitrary persons should be verifiable, without requiring a large database of each person's signatures that are used for comparison. These requirements make it challenging to create a robust and accurate automatic signature verification system.

Meanwhile, new advances in the field of machine learning, particularly deep learning, make it worthwhile to consider the application of such approaches to the problem. To date, this has only been attempted to a limited extent [24].

In this thesis, we propose a new approach to automatic signature verification that is based on modern techniques of deep learning and addresses the challenges above. We call our approach *Signature Embedding*.

The remainder of this chapter discusses signature verification in the context of biometric authentication and provides an overview to the contributions and the structure of this thesis.

1.1 Signatures as a Means of Biometric Authentication

The verification of handwritten signatures is a form of *biometric authentication*. In other words, it is a means of authentication that is based on a person's inherent biometric traits, or parameters.

Biometric parameters are one of three authentication *factors* that are commonly distinguished [13]. The other two factors are:

KNOWLEDGE, where identity is proven by something a person knows, for example a password or a PIN.

OWNERSHIP, where identity is proven by something a person possesses, for example a physical key or a digital key (certificate).

Multiple factors can be combined in order to provide higher security. A commonly employed scheme is requiring both the knowledge of a password and the possession of a mobile phone with a specific phone number¹. Analogously, the ability to produce the requested signature can be combined with the requirement to possess a matching identity card.

Besides the handwritten signature, a plenitude of other biometric parameters have been employed for authentication. They are classified according to two sub-categories: *physical* and *behavioral* parameters [33]. Physical parameters include the fingerprint, facial features, and the DNA. Behavioral parameters include a persons gait, the rhythm of keystrokes, and the handwritten signature.

A biometric authentication system can operate in one of two modes, depending on its task. These two modes are *identification* and *verification* [33]. In the identification mode the task is to answer the question which known user of the system a set of biometric parameters belongs to. In the verification mode the task is to determine whether or not a set of biometric parameters belongs to a certain, specified user.

An additional *enrollment phase* may be required. In this phase users and their biometric parameters are registered to the system's database.

1.1.1 Automatic Signature Verification

In this thesis we are concerned with the use of signatures for the purpose of verification, more specifically, with the software required in this process in order to verify digitalized signatures. The hardware employed for capturing signatures is not in the focus of this thesis.

Given the framing scenario we consider, the system we propose in this work will be facing a number of challenges. As noted above, we assume a scenario where signatures are provided without any additional information about the process of their creation, such as the pressure exerted on the pen. Consequently, the signatures on which our system operates are static images. This scenario is termed *off-line* signature verification.

Another requirement we formulated above is that no large database of users' signatures is available. This concerns the aforementioned enrollment phase, which is thus omitted in the system. Systems that operate without a database of their users' signatures are termed *writer independent* signature verification systems.

¹ One instance of this scheme is implemented by Google in order to protect users' on-line accounts from unauthorized access (<https://www.google.com/landing/2step/>, accessed 01.08.2016)

Finally, all signature verification systems need to handle the fact that the handwritten signature of a person can change over time. This is a challenge that most biometric authentication systems are facing. However, in contrast many other biometric traits, the signature of a person is known to vary substantially even across two successive impressions [33]. The taxonomy of signature verification tasks and the challenges arising for them are discussed in detail in [chapter 2](#).

1.1.2 Ethical Considerations

As noted above, the handwritten signature is used as a means of authentication in various areas of our life. This gives rise to the need of ethical considerations of biometric authentication systems based on signatures.

By their nature, biometric parameters are suitable as a means to identify a person. This makes them a personal quality and subject to considerations of privacy and informational self-determination. As a result of the increased availability and employment of biometric authentication in both consumer applications and for governmental purposes, various reports have addressed these questions (e.g. [12, 20, 49]).

In a 2004 report on biometric-based technologies issued by the Organisation for Economic Co-operation and Development (OECD) [12], three areas of concern were identified: First, the data that is gathered by biometric systems gives rise to concerns about *function creep*—the extension of a system’s area of application beyond the task it was originally designed and deployed for. Second, the extensive deployment of biometric identification systems carries the “risk that these systems may become an infrastructure of surveillance” [12]. Third, implementations of such systems might not guarantee transparency and users’ consent to their employment.

In consequence, the report provides a number of recommendations about the design of biometric authentication systems. Among other points, it is emphasized that

- biometric *verification* systems carry a smaller risk to privacy than biometric *identification* systems
- biometric samples should be collected openly and with the consent of the user
- users should retain custody of their biometric data and data should not be stored in a centralized database

These concerns and recommendations apply to automatic signature verification systems as well. It should be noted that the handwritten signature is, in some respects, not as crucial as other biometric parameters. A person’s facial features, for example, could be captured by video surveillance in public places without the person noticing. The signature, in contrast, has to be actively created by the person. Furthermore, the collection of signatures is often perceived as less invasive, compared to other biometric parameters, such as the fingerprint [58].

Nonetheless, a number of measures needs to be considered, especially for systems deployed at large scale. The Advisory Body on Data Protection and Privacy of

the European Commission [20] put forth the *purpose principle*, stating that personal data should only be collected for explicit and legitimate purposes. Applying this principle to signature verification systems, this means that the signature should be used exclusively for the purpose that it is explicitly collected for. This relates to the aforementioned risk of function creep.

Furthermore, the Advisory Body, too, expresses that data should not be stored in a centralized database and that “unnecessary data should be destroyed as soon as possible” [20]. In this respect, writer independent systems are favorable, as they do not rely on a database of users’ signatures at all. Instead, they are provided with a number of known genuine signatures of the user in question at the time of their usage. No user specific data needs to be stored.

1.2 Contributions

In this thesis we present a new approach to writer independent off-line signature verification. The system we created is based on deep learning technologies. It operates on static signature images and does not rely on users to be enrolled in a database. The contributions of this thesis can be summarized as follows:

- We investigated how deep learning techniques can be employed to the task of writer independent off-line signature verification. In this domain, the use of deep neural networks (DNNs) has previously not been widely researched [24]. We establish how deep metric learning can be used for this task.
- We implemented a system that makes use of our approach. The system is implemented with the help of the widely employed deep learning framework *Chainer*² [74]. Using the building blocks provided by Chainer, we implemented a specific loss function (*triplet loss*) that was used to train our DNN.
- We contributed two pull requests to the Chainer project^{3,4}. The pull requests add a mathematical function that can be used as part of the computations within a neural network, and an efficient, GPU-based implementation of that function.
- In order to allow for best reproducibility of our results, we created a Python⁵ package that implements the computation of the aforementioned triplet loss function. Both this Python package and the rest of the source code for this project are openly available⁶.
- We explored metrics to validate a system like ours while it is trained and to evaluate its performance when it is used. Moreover we investigated datasets that can be employed for training and evaluating the system.
- We evaluate the performance of our system and compare it to the results of the ICDAR SigWiComp 2013 [44] challenge on off-line signature verification. Our

² <http://chainer.org> (accessed 20.07.2016)

³ <https://github.com/pfnet/chainer/pull/828> (accessed 07.08.2016)

⁴ <https://github.com/pfnet/chainer/pull/871> (accessed 07.08.2016)

⁵ <https://www.python.org/> (accessed 20.07.2016)

⁶ <http://hannesrantzsch.de/projects/signature-embedding>

results compare favorably, ranking best with regards to all employed metrics on one of two tasks, and in most of the employed metrics for the other task.

In the course of this evaluation, we establish that our approach is generalizable enough to operate even on signatures in scripts that it has not been trained on. Specifically, the system performs well on the verification of Japanese script signatures, even though it is trained exclusively on Latin script signatures.

1.3 Outline of the Thesis

This thesis is organized as follows: In [chapter 2](#) we introduce the reader to the domain of signature verification and the signature verification problem. We give an overview of various scenarios that can be addressed as sub-tasks of this problem and define the specific scenario addressed in this work. Subsequently, in [chapter 3](#), we provide an introduction to the technological background relevant in this thesis. Chapter [4](#) names and explains the data, hardware, and software resources we employed in this project.

Chapter [5](#) conveys an intuition to our approach and discusses how the system can be created and used. Subsequently, [chapter 6](#) details on the implementation of the system: the design of the neural network, the preparation of data, and the training of the model.

In [chapter 7](#) we evaluate the performance of our system and compare it to the results produced by other systems. Furthermore, we experimentally assess the implications of a number of design decisions we took.

Chapter [8](#) reviews our approach in the context of related work on signature verification. We conclude with a summary of the content in this thesis in [chapter 9](#). Furthermore, we discuss insights we took from this project, as well as possible future improvements and applications of our approach.

2 The Signature Verification Problem

In this chapter we first provide an introduction to the domain of signature verification and to writer independent off-line signature verification systems in particular. Subsequently, we specify the exact problem addressed by this work.

2.1 Signature Verification

The task of *signature verification* describes the problem to confirm or to refute the genuineness of a signature; in other words, to verify whether or not the signature was created by a certain person. If the signature was not created by that person, the signature is either a genuine signature of another person or has been forged on purpose.

2.1.1 Types of Forgeries

The exact reason why the signature at hand is not the genuine signature of the person in question is not relevant for our task of verifying it. We will thus use the term *forgery* in both of the cases named above. In fact, most signature verification literature distinguishes three kinds of forgeries [24], depending on how much knowledge a forger has about the person whose signature they attempt to imitate:

RANDOM FORGERIES Here the forger knows neither the signature nor the name of the person they try to impersonate. These forgeries can easily be recognized, as the forged signature bears no resemblance to the genuine signature.

SIMPLE FORGERIES In this case the forger knows only the other person's name. This kind of forgeries can be more similar to the genuine signature, especially if the person signs with his or her full name.

SKILLED FORGERIES In this scenario the forger has knowledge about both the person's name and signature. The forger also has the possibility to practice the imitation of the signature. Thus, it can be very hard to distinguish the forgery from a genuine signature.

In order to approach and solve the problem of signature verification, a signature verification system depends on the information available in the same way that a forger does when crafting a hard to detect forgery. Generally speaking, detecting a forgery requires to have more knowledge about the genuine signature than the forger has. Given no information about the signature the forger strives to imitate, there is no basis on which to decide whether a signature is genuine or forged. Given only the person's name, only random forgeries can be detected. In contrast,

detecting a skilled forgery requires at the very least one signature that is known to be genuine, so it can be compared to the potential forgery.

2.1.2 On-Line and Off-Line Signature Verification

Depending on the information at hand about the signing process, signature verification is categorized as *on-line* (or dynamic) or *off-line* (or static) signature verification [58]. In the on-line scenario information about the process of creating the signature is available. The type of this information depends on the device that is used to capture the signing process. Data is captured as a function of time, resulting in a sequence of input data points. Data captured in such a process always includes the position of the pen at each capture point. In addition, it can include information such as inclination and pressure. It is very hard to accurately imitate such information, which makes the detection of forgeries easier.

For signatures we encounter in our everyday life such information is usually not available. Signatures written or printed on paper are static images. The scenario where only static images of both genuine signature and potential forgery are given is termed off-line signature verification.

This thesis is concerned with the design and implementation of an off-line signature verification system.

2.1.3 Off-Line Signature Verification

First of all, it is important to note that the problem of “off-line signature verification” can be defined in different ways. In fact, works in literature often diverge in their definition of the task.

The first differentiation to be made is whether or not skilled forgeries are used. In contrast to a signature identification system, a signature verification system always needs to be able to detect skilled forgeries. Nonetheless, not all approaches make use of samples of skilled forgeries in order to build the system.

The second differentiation is the notion of *writer dependence* and *writer independence*.

Writer Dependence and Writer Independence

Whether a signature verification system is considered writer dependent or independent is determined by the group of persons whose signatures it is able to verify. We call this group of persons *users* of the system. A signature whose genuineness is to be verified by the system is termed the *questioned signature*, or questioned sample.

A writer dependent system is able to verify signatures of known users only. Such a system relies on users to be enrolled when it is built. As these systems are typically machine learning based, this means the system is trained on sample signatures of the enrolled users. As a result, training these systems can require a large number of sample signatures of each user, which is inconvenient to acquire.

In return, using writer dependent systems is most convenient. The input to the system is a single questioned sample, which the system can classify as genuine or forged based on its knowledge about its users.

Writer independent systems, in contrast, are able to verify signatures of previously unseen users. However, in addition to a questioned sample they require one or more *reference samples* as input. These reference samples are compared to the questioned sample.

In order to do so, the system needs to extract an abstract, comparable representation from the input samples. Writer independent signature verification system hence often rely on *similarity representations* of signatures, which allow them to estimate two signatures' similarity. Subsequently, the system decides whether the questioned signature is genuine or forged based on this comparison.

Challenges

The task of off-line signature verification has a number of properties that introduce specific challenges. One issue is a very high intra-class variability [24]. In other words, the visual difference between two genuine signature of the same user can be quite large. At the same time the visual difference of a skilled forgery to any of those signature is only marginally larger. This makes the task very hard, especially for writer independent systems, which do not possess much information about the individual user. Figure 2.1 exemplifies the large intra-class variability among one user's genuine signatures.



Figure 2.1: Superimposed genuine signatures of one user show a large intra-class variability. Image from [35].

Another issue is the availability of training data. This is a problem especially for writer dependent systems that rely on the enrollment of users. If systems are designed to be employed in real world applications, users usually cannot be bothered to provide a large number of signature samples for the system to be trained with. Writer independent systems in contrast can rely on existing databases for training.

Unfortunately the availability of training data for off-line signature verification falls short in comparison to other computer vision tasks in two respects. The first problem is the size of the databases. A widely used image database for image recognition is ImageNet [15]. ImageNet provides more than 14 million images for more than 21 000 classes with around 500-1000 images per class¹. These images

¹<http://www.image-net.org/about-stats> (accessed 15.06.2016)

are automatically crawled from the Internet, but quality controlled and human-annotated². Mapping users to classes and images per class to signatures per user, it would be an enormous effort to collect that many signatures from that many people.

The second problem lies in obtaining existing databases. A database that has been widely used in literature ([2, 51, 83]) is the “GPDS-960 corpus”[79]. Unfortunately, at the point of writing this thesis, this database is no longer available due to licensing issues³. Various other databases, for example ones used in the ICDAR SigWiComp Challenges for signature verification from the years of 2011, 2013, and 2015 ([40, 44, 43]), are either not at all or only partly available at this point.

Finally, we want to consider the notion of an adversary with unlimited resources. This consideration is common in the IT security community, which is related to the topic of signature verification, if signatures are to be used as a means of authentication. Applied to the scenario of off-line signature verification, this means a forger has the knowledge about the genuine signature, unlimited time, and the capability to create a perfect forgery. For an off-line signature verification system it is not possible to detect such a forgery. As a result, we need to rely on the fact that a forger can only create forgeries of limited quality.

2.2 Task Specification in This Thesis

The system proposed in this work is a writer independent system for off-line signature verification. We make use of skilled forgeries for training the system. The system’s performance is evaluated on a set of users whose signatures are not part of the training set. In our evaluation (chapter 7) we use a small number of reference signatures in order to make a decision concerning the genuineness of a questioned signature. The scenario where only one reference signature is available is also discussed (section 7.5).

²<http://www.image-net.org/about-overview> (accessed 15.06.2016)

³<http://www.gpds.ulpgc.es/> (accessed 15.06.2016)

3 Theoretical Background

Since our system is based on machine learning, and more specifically deep learning, we want to introduce the basic concepts of these technologies in this chapter. Furthermore, we will examine similarity metrics in the context of machine learning.

During this discourse over the relevant technologies, we will also mention the corresponding literature. Note however, that the related work in terms of signature verification is discussed in [chapter 8](#).

3.1 Machine Learning

Machine learning describes a field of computer science that is concerned with algorithms that *learn* to solve a task based on observations, rather than being explicitly programmed how to solve the task. Machine learning approaches are popular for example in the field of computer vision. Typical applications for machine learning in this field include recognizing what is shown on a picture (image recognition) [38] and generating a natural language description for it (image captioning) [80], or reading text in a picture (optical character recognition) [32].

3.1.1 Types of Machine Learning

Machine learning is commonly divided into three sub-fields, depending on the amount and type of *feedback* that is available to the algorithm while it learns [64]:

SUPERVISED LEARNING, where input data is labeled and the algorithm is provided with feedback whether or not its output corresponds to the expected label. support vector machines (SVMs) as well as the artificial neural networks discussed below are typical examples for supervised learning.

UNSUPERVISED LEARNING, where labeled data is not provided to the algorithm. Data clustering techniques, such as k-means [42] and Gaussian mixture models, are a kind of unsupervised learning.

REINFORCEMENT LEARNING, where the algorithm is not provided with the expected label for an input, but with *some* kind of evaluation, how good its output was. *Q-learning* is a kind of reinforcement learning that gained a lot of attention in the recent years and was successfully applied to a variety of problems [48].

Mixtures of these types, such as semi-supervised learning, can also be found. The system presented in this thesis relies on supervised learning techniques.

3.1.2 Supervised Learning: Linear Regression

As a simple example of supervised learning, consider linear regression. Linear regression describes the task to *model* the dependency between an input vector $x \in \mathbb{R}^n$ and a scalar output $y \in \mathbb{R}$ [3]. In other words, the algorithm should be able to predict the value of y for a given input *sample* x .

In order to approximate y , the output of the model is defined as

$$\hat{y} = w^T x + b, \quad (3.1)$$

where \hat{y} is the model's prediction for y . $w \in \mathbb{R}^n$ is termed the model's *weights* and $b \in \mathbb{R}$ its *bias*. When the model learns, weights and bias are adjusted to improve the approximation \hat{y} .

w and b are summarized as the model's *parameters*. Note that in some literature the term weights is used to describe both w and b . This can be explained by the fact that b can be emulated as part of the weights. In order to do so, both x and w are augmented by an extra entry, which is always set to 1 in x . In this thesis, however, we will use the term parameters to refer to both weights and biases.

The process of adjusting the parameters to improve the approximation is called *training* the model. The training is based on a given set of *labeled samples*, whose corresponding value in the target function are known. The prediction \hat{y} produced by the model and the expected y can hence be compared in order to adjust the parameters. This set of labeled samples is termed the *training set*.

In addition to the training set, a test set is employed to validate the model's generality. This test set contains labeled samples as well, but is not used for updating the model's parameters. It can thus be used to verify whether or not \hat{y} is a good approximation to y , even on samples the parameters have not been explicitly adjusted for.

Underfitting and Overfitting

In addition to finding good parameter values, another dimension to the problem is choosing the right model. The problem of linear regression above can be generalized to the problem of polynomial regression. As the name suggests, rather than a linear function, a polynomial function is employed in order to approximate y . Our prediction for a polynomial of degree n is hence defined as

$$\hat{y} = \sum_{i=0}^n w_i x^i + b. \quad (3.2)$$

It is unknown how the degree of the polynomial n should be chosen in order that \hat{y} approximates y best.

However, choosing n well is of great importance. If n is chosen badly, no good approximation of y will be found, regardless how well the parameters are adjusted. The choice of the degree can affect the model in the following way:

Choosing the degree too low results in the model's inability to reproduce the target function. This effect is referred to as *underfitting*. Underfitting reflects in

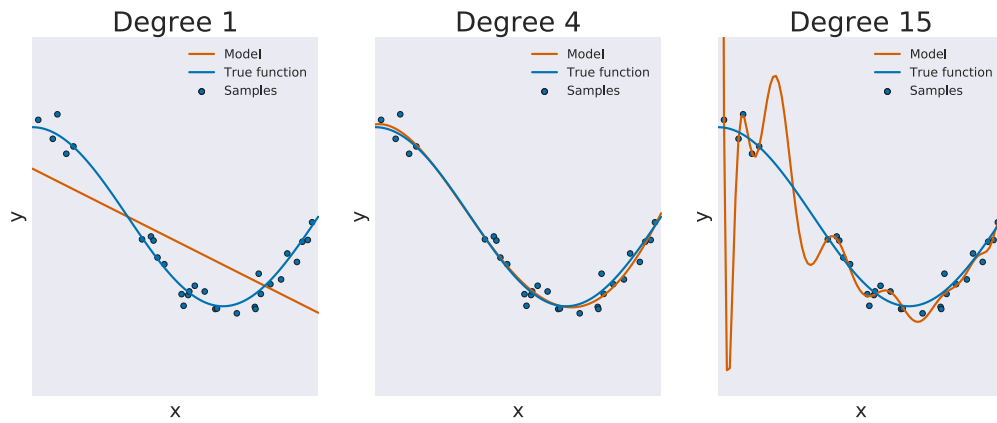


Figure 3.1: Choosing the degree well is crucial in polynomial regression. A too small degree leads to underfitting (left), while a too high degree causes overfitting (right).¹

both the training set and the test set in a way that the model is not able to produce good approximations.

The opposite effect can occur as well: If the degree is chosen very high, \hat{y} might be able to approximate y for data in the training set extremely well. However, the training set is always just a subset of the real data and can include outliers in any real world scenario. It is hence possible that a simpler function exists that does not approximate y on the training data quite as well, but performs much better on unseen data. This effect is called *overfitting*. Overfitting thus causes the model to produce very good approximations on the training set, but very poor approximations on the test set. Underfitting and overfitting are illustrated in figure 3.1.

The resulting trade-off between facilitating a good approximation on the training set while maintaining generality is also referred to as the *bias-variance tradeoff* [22].

3.1.3 Classification

Many machine learning tasks, such as the image recognition mentioned in the beginning of this section, are classification tasks. This means the algorithm is given an input sample and is supposed to output the correct label—or *class*—for this sample (for example the class “cat” for a picture showing a cat).

Note that in classical machine learning—that is, not deep learning—the input for the algorithm is not the raw input sample. Instead, the input sample needs to be preprocessed and *features* need to be extracted. The features then form the input to the classification algorithm.

In contrast to the linear regression task described above, whose output was a scalar variable, classification tasks require the algorithm to predict discrete or categorical variables. The framing setup, however, remains the same. The algorithm is provided with labeled training data and should adjust parameters accordingly.

¹Adapted from http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html (accessed 01.07.2016)

3.1.4 The Loss Function

We now want to examine how a prediction \hat{y} can be evaluated in order to adjust the parameters. Therefore, a measure of how well \hat{y} approximates y is required. Alternatively, a measure of error, in other words, how far the approximation is from the expected output, can be employed. Such a measure of error is termed *loss function*.

A very simple loss function is *mean squared error* (MSE). It is defined as

$$L_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad (3.3)$$

and would be very suitable for the aforementioned linear regression problem.

For classification problems there are no continuous values y and \hat{y} to compare. The predicted class is either the correct one or a wrong one. Furthermore, multiple possible classes have to be taken into account. On a given input \mathbf{x} , a multi-class classifier thus typically assigns a vector of unnormalized probabilities $P_k(y = k \mid \mathbf{x}; \mathbf{w}; b)$. The value in this vector at index k denotes the probability the classifier estimates for \mathbf{x} being of class k .

A commonly applied loss function for such scenarios is *cross-entropy loss*, which is defined as

$$L_{\text{cross-entropy}} = -\log \left(\frac{e^{f_y}}{\sum_k e^{f_k}} \right), \quad (3.4)$$

where f_y is the probability granted for the correct class and the f_k are the probabilities for all classes. Part of the definition of cross-entropy loss is the function

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}. \quad (3.5)$$

In equation 3.4, $\sigma(f_y)$ can be interpreted as the normalized probability assigned to the correct label y . $\sigma(f_y)$ is called the *softmax* function. Classifiers employing cross-entropy loss are accordingly called softmax classifiers.

3.1.5 Adjusting the Parameters

An objective loss function, that can be employed as a measure of how good the current model is, allows us to understand the problem of finding good parameter values as an optimization problem. Target of the optimization is minimizing the loss function. One such optimization method, that is of particular importance in deep learning [3], is *stochastic gradient descent* (SGD), an extension of gradient descent. Gradient descent was first formalized in 1847 by Augustin Cauchy [9]. We will provide only a brief overview of the mathematical concepts involved here.

Suppose the target of optimization is the continuous, differentiable function $L(\phi)$, calculating the loss for given parameters ϕ . Gradient descent makes use of the first order derivative $L'(\phi)$ in order to minimize $L(\phi)$. $L'(\phi)$ gives the slope

of $L(\phi)$ at ϕ , and can thus give insight into how a small change ϵ in ϕ reflects in $L(\phi)$:

$$L(\phi + \epsilon) \approx L(\phi) + \epsilon L'(\phi). \quad (3.6)$$

It follows that

$$L(\phi - \epsilon \text{sign}(L'(\phi))) < L(\phi) \quad (3.7)$$

for small enough ϵ [3].

Loss functions above operate on an input vector $\phi \in \mathbb{R}^n$. In order to minimize these functions the concept of *partial derivatives* is employed. $\frac{\partial}{\partial \phi_i} L(\phi)$ characterizes how L changes as only ϕ_i changes. The *gradient* of L , denoted $\nabla_\phi L(\phi)$, is the vector containing all partial derivatives of $L(\phi)$. The method of *steepest gradient descent* is then used to propose ϕ' , such that $L(\phi') < L(\phi)$ as

$$\phi' = \phi - \epsilon \nabla_\phi L(\phi). \quad (3.8)$$

Stochastic Gradient Descent

Of course the loss depends on the input as well. So far we have always considered the loss for individual samples. But a better estimation of how to descend the gradient—and thus how to adjust the parameters—can be obtained by considering all the samples in the training set. In order to adjust the model's parameters based on multiple labeled input samples, the loss for each of the samples is calculated and the appropriate parameter update for each individual sample is computed using the gradient descent method. Subsequently, the mean of all parameter updates is calculated and applied.

Depending on the model and the loss function, obtaining the output \hat{y} and calculating its loss can be computationally expensive. Machine learning, and particularly deep learning, often requires a large training set. There is hence a limit to how many samples from this set can be considered for a single step of descending the gradient. This issue is addressed by the aforementioned SGD method. The key insight of SGD is that a step on the gradient descent can be approximated using a smaller set of samples, uniformly drawn from the training set [3]. These smaller sets of samples are called *batches*, or sometimes *minibatches*.

The number of samples in each batch, the *batch size*, is typically fixed and chosen initially to the training, specific to the system and data at hand. Commonly, the batch size lies in the range of 1 to a few hundred samples [3].

Each parameter update is thus computed based on one batch. Let ϕ be the current parameters of the model and g_{batch} the estimated gradient based on a batch. Then, analogously to equation 3.8, the next step descending the gradient is

$$\phi' = \phi - \epsilon g_{batch}. \quad (3.9)$$

ϵ is termed the *learning rate*. The learning rate hence scales the parameter update by a constant factor. This is particularly necessary when approaching a the minimum.

If the step is too big, the minimum could be missed. On the other hand the learning rate should not be chosen too small, especially in the beginning, since it obviously takes longer to descend the gradient in smaller steps. The learning rate is thus typically decreased as the training proceeds.

Other Update Methods

SGD is not the only method that can be employed for updating the model's parameters. It is, however, the foundation which many other methods are based upon. A fairly simple extension of SGD is *momentum* [63], or momentum SGD. Intuitively, momentum speeds learning by accumulating a moving average over past gradient updates and fostering movement into this direction. Therefore, the velocity variable v is added to SGD [3]. v is used to keep track of past parameter updates and accelerate progress along dimensions in which the gradient consistently points to the same direction, while slowing progress in dimensions where the sign of the gradient continuously changes [84]. In addition, v decays over time. Momentum is also employed in the system proposed in this thesis.

Other gradient descent optimization methods that are commonly employed¹ include *Nesterov momentum* [4], *Adagrad* [18], *Adadelta* [84], *RMSprop* [72], and *Adam* [37].

3.2 Artificial Neural Networks

As the name suggests, artificial neural networks (ANNs) are inspired by (biological) neural networks, in particular the animal brain. Neural networks are made of interconnected neurons. Analogously, ANNs are made of interconnected, artificial neurons.

One of the first formal descriptions of an artificial neuron is attributed to McCulloch and Pitts [46]. In 1943, they proposed a mathematical model of a neuron that would “fire” if its weighted inputs overcome a defined threshold. In order for the artificial neuron to produce the desired function, the weights needed to be set correctly by a human operator.

3.2.1 The Perceptron

Inspired by McCulloch and Pitts's artificial neuron [52], Rosenblatt invented the *perceptron* [61]. The perceptron produces a binary output based on a vector of weighted inputs and a threshold θ :

$$output = \begin{cases} 1 & \text{if } w^T x > \theta \\ 0 & \text{otherwise} \end{cases} \quad (3.10)$$

¹ All of these methods, as well as SGD and momentum, are found in major deep learning frameworks, such as Caffe (<http://caffe.berkeleyvision.org/>), Chainer (<http://chainer.org/>), and Tensorflow (<https://www.tensorflow.org/>).

In contrast to McCulloch and Pitts's artificial neuron, the perceptron's weights are adjusted in a learning process. We have already discussed methods to update parameters. These methods can be applied to the perceptron as well. A simple perceptron with three inputs is illustrated in figure 3.2. In this figure, the threshold that determines the perceptron's output is depicted as a step function.

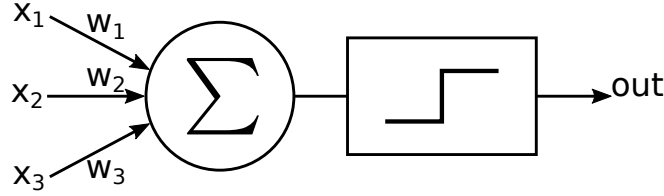


Figure 3.2: The perceptron's output is determined by a step function over the sum of its weighted inputs.

3.2.2 The Multilayer Perceptron

Due to its simplicity, the perceptron is quite limited regarding what functions can be modeled with it. Most famously, it cannot learn to represent exclusive OR (XOR) [47, 3], which is not linearly separable.

This limitation is overcome by the multilayer perceptron (MLP). As the name suggests, the MLP consists of multiple perceptrons, each representing one neuron in the neural network. The perceptrons are arranged in *layers*. An example of an MLP is depicted in figure 3.3. The individual perceptrons in the MLP are interconnected in a way that the output of each layer of perceptrons forms the input to the next layer.

The MLP in this example consists of three layers, the second of which is termed *hidden layer*. The hidden layer is neither an input nor the output layer. Nonetheless, it contributes to determining the output. Its input is the output of the first layer. The two layers can thus abstract from the “raw” input features, which allows the MLP to represent more complex functions (such as XOR). MLPs can have more than one hidden layer.

The MLP depicted in figure 3.3 has only one output neuron and thus performs a binary classification, producing 0 or 1. Employing multiple output units allows for multi-class classification. In this case, one output unit would be employed for each class, responding 1 if the respective class is detected and 0 otherwise.

The Sigmoid Function

In section 3.1.5 we introduced gradient descent, which allows for adjusting the model's parameters. The basis for this is the observation of how small changes in the input reflect in the output. More formally, the gradient is computed based on the vector of partial derivatives of the loss function L in x

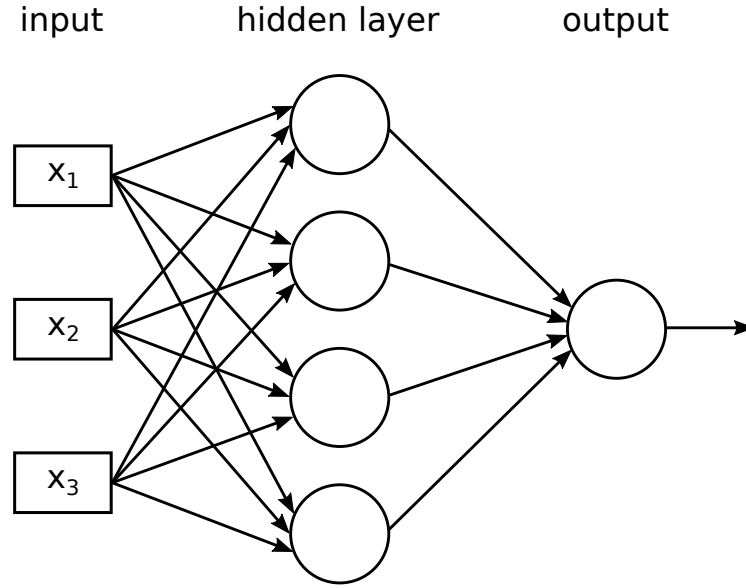


Figure 3.3: The MLP is a network of interconnected perceptrons. In this figure, each circle indicates a perceptron.

The step function within the perceptron, however, is not differentiable in θ . Thus, the *sigmoid function* is employed instead. It is defined as

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \quad (3.11)$$

3.2.3 Adjusting the Parameters

The parameters in ANNs can be updated using SGD and methods derived from it, just like described in section 3.1.5. However, ANNs consist of multiple layers of neurons, which complicates the computation of the gradient.

In order to obtain the gradient, the *back-propagation* [63] algorithm is employed. We will not discuss the back-propagation algorithm in detail. Briefly explained, back-propagation makes use of the chain rule of calculus. Based on the loss and starting with the final output layer, an *error* is calculated and propagated backwards to the previous layer, in order to obtain the error for this layer in turn. Once the errors are computed for each layer, they can be used to calculate the gradient of the loss function.

Analogously to the back-propagation, the operation of layer-wise processing of the network's input in order to obtain the output and the loss is termed *forward pass*.

3.3 Deep Neural Networks

There is no unambiguous definition for the terms *deep neural network* and *deep learning* [3]. Most concepts in deep learning have existed in the field of machine

learning since the 1940s. At that point, the name deep learning had not yet been coined—instead the field was called *cybernetics* or, later, *connectionism* [3].

Deep neural networks (DNNs) and deep learning usually refer to the recent advent of these technologies, beginning with Hinton et al.’s publication in 2006 [26], suggesting a fast, greedy learning algorithm that can train a model layer-wise. The modern approaches to DNNs are characterized by a larger number of layers, comparing to the ANNs described above, as well as by a greater variety of layer types. Perhaps the layer type that is most characteristic to DNNs is the convolutional layer.

In the following we will briefly discuss operations and layers that form the major building blocks of DNNs—such as the convolutional layer—and that are of particular relevance in this thesis.

3.3.1 Building Blocks of Deep Neural Networks

The Fully Connected Layer

The layers of perceptrons that formed the hidden layers in MLPs are still relevant in DNNs. However, they are usually entitled differently. One name for this type of layer is *fully connected* (FC) layer, another commonly used name is *linear* layer.

In an FC layer all outputs of the previous layer are connected to all inputs of the FC layer—hence the name. Typically in DNNs, as well as in the network employed in this work, one or more instances of this type of layer are used as the final layers of the network, performing a feature-based classification task, just like they do in MLPs.

The Convolutional Layer

The type of layer that typically provides the features to the FC layers is the *convolutional layer*. This layer earns its name by the operation it performs: the convolution. Refer to [figure 3.4](#) for an illustration of the convolution.

Convolution is usually a two-dimensional operation. This means, in contrast to the FC layer it does not operate on the complete input as a *flat* vector, but stepwise processes two-dimensional patches of the input. This is easiest exemplified with an image as input. In this case, the convolution processes the image as follows:

The most important part of the convolution is its *kernel* (sometimes called *convolution matrix*). The kernel is a matrix of size $k_w \times k_h$, containing $k_w \times k_h$ weights that are adjusted during training.

The output of a convolution can be interpreted as an image as well. The kernel with its weights determines the value of one pixel in the output image. Therefore, all pixels in a $k_w \times k_h$ -sized patch of the input image are multiplied with the respective weight, and subsequently summed. Additionally, the convolutional layer can learn a bias to the kernel, which is added to the result.

This determines the value of one pixel in the output image. In order to process the whole input image, the kernel is *slided* across the input image. This means the kernel is moved to the next $k_w \times k_h$ -sized patch in the input image, multiplying

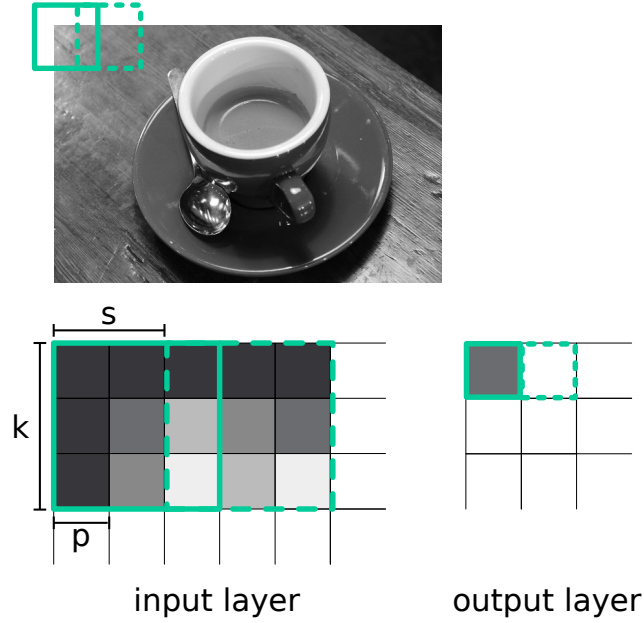


Figure 3.4: The convolution is slid over the image in the input layer, stepwise producing the output. This figure exemplifies a convolution with *kernel size* $k = 3$, *stride* $s = 2$, and *padding width* $p = 1$. The dashed square indicates the next processing step.

and summing the pixels found there with the same weights and bias used before. Thus, the second pixel of the output image is created. This process continues until the whole input image has been processed.

How the kernel is slid across the image is determined by an additional parameter called *stride*. For example, a stride of 3×5 means the kernel is moved by three pixels in one dimension (width) and five pixels in the other (height). Another parameter that needs to be provided to the convolution is the *padding width*. Padding can be added at the borders of the input image, in case kernel and stride do not fit the input size and the kernel would include non-existent pixels.

The last parameter of the convolution that shall be relevant for our description here is the number of *filters*, or filter dimensions. This parameter determines *how many kernels* the convolution slides across the image. All these kernels are moved across the image as described above and have the same kernel size. However, the weights and bias they learn differ. One convolution is thus able to produce multiple output images from one input image. As an example, consider a convolution with 64 filters applied to a grayscale input image. This convolution would then slide 64 kernels with same size, padding and stride across the image, learning different weights and bias for each of the 64 kernels; hence producing 64 output images. Analogously, a convolution with 64 filters could be applied to an RGB color image. The input image thus already has three dimensions (red, green, and blue) to each of which the 64 filters are applied, resulting in $3 \cdot 64 = 192$ output images.

Conceptually, the convolutional layer can be understood as a feature extraction. For example, the layer could learn to perform edge detection using a Sobel filter, which is a specific kernel of size 3×3 that can be employed for detecting edges [69].

This convolutional layer could be followed by further convolutional layers, that in turn extract more abstract features, for instance shapes, from the output of the edge detection.

This property makes convolutional layers a very characteristic part of DNNs: In DNNs, the extraction of features from samples is often no longer engineered manually, but automatically learned utilizing convolutional layers.

Further Layers and Components

DETECTOR Just like the step function and the sigmoid function employed in MLPs, FC and convolutional layers need an activation function as well. The aforementioned sigmoid function is a valid choice for this task. In some cases though, it has been shown that DNNs can converge faster if another function is used: the Rectified Linear Unit (RELU) [50]. RELUs are simply defined as lower-bounding the output to zero.

POOLING Oftentimes, convolutional layers are followed by a special *pooling* layer. Pooling layers behave very similar to convolutions, in that they slide an operation across an input. In contrast to convolutions though, pooling layers do not learn parameters. Instead, they simply average over the patch (*average pooling*), or take the maximum value in the patch as the resulting output value (*max pooling* [86]). Typically, pooling layers fulfill two main tasks: they reduce the size of the input and make the resulting representations more translation invariant [3].

A Note on Terminology

Which of the layers and components detailed above are used in the DNN and how they are combined determines whether or not the network will be able to fulfill its purpose. We refer to this as the *layout* or *architecture* of the network. In addition to these terms, there is the term of the (neural network) *model*. With the term *model* we refer to a specific instantiation of the neural network. In other words, *layout* describes which layers are employed, while the *model* is set up according to the *layout*, trained, and ultimately used to make predictions.

3.3.2 DNNs and GPUs

In the beginning of this section it was mentioned that deep learning technologies gained new popularity in recent years. An additional factor that contributed to this effect is the increased computational power of modern computer hardware. Even though we stated that many of the algorithms employed in DNNs already existed in the middle of the 20th century; the hardware available at that point was not powerful enough allow for the creation of truly *deep* networks with a large number of network layers.

This trend was further fostered by the rise of graphics processing units (GPUs) as a general-purpose computation device [56]. Many instructions, such as the matrix multiplication that is part of the convolution, as well as the update of

the parameters using the computed gradient, are very suitable for the parallel capabilities of the GPU.

This tendency was also noticed by the GPU vendor NVIDIA², who thus provides a now widely adopted “library of efficient implementations of deep learning primitives” [10], called cuDNN³.

3.4 Similarity Metrics

The final part of this technology overview is concerned with similarity metrics, and how a similarity metric can be found for given data.

In other words, the goal is to find a means that allows for a metric comparison of the data. We will assume images as input data here, as signatures are images, and similarity metrics, particularly in the context of machine learning, are mostly used for images.

The main requirement is that the comparison reflects the *semantic similarity* of the images (note that the semantic similarity needs to be defined by the concrete use case). In order to allow for such a comparison, the input data is mapped (or *embedded*) into a space where this comparison is possible. An example is the Euclidean space. Here, the Euclidean distance provides a metric that can be employed for comparison.

The challenge is finding a function that performs this embedding in a way, that the desired probabilities for the comparison are fulfilled. Suppose this function f_w , parameterized by weights w , performs an embedding of an input x into Euclidean space. Then a formal definition of the similarity metric M is given by

$$M_w(x_1, x_2) = \|f_w(x_1) - f_w(x_2)\|_2. \quad (3.12)$$

The requirement to M_w is that it is small, if x_1 and x_2 are semantically similar, and large if they are dissimilar.

3.4.1 Similarity Metrics and Machine Learning

Similarity metrics have found many applications in machine learning. For instance, Turk and Pentland performed a principal component analysis (PCA) in order to extract eigenvectors from face images [75]. In this context, the PCA is used as an embedding function. The resulting *eigenfaces* were used as input features for training an MLP.

This is, however, only one way to combine the two concepts. In the description above, the embedding function was given as a function f_w , parameterized by weights. Another way to combine similarity metrics with machine learning is using neural networks to find the function’s parameters, in other words, to learn the embedding function.

²<http://www.nvidia.com>

³<https://developer.nvidia.com/cudnn>

This was demonstrated by Bromley et al. in 1993 [6]. The authors used two neural network models, sharing the same parameters and hence computing the same function. Each model was used to embed the (handcrafted) feature vector of one labeled sample into Euclidean space. If the two samples belonged to the same class, the models should embed both of the samples nearby each other. This method provides an way to easily compare two feature vectors. Since the two models they used are identical, the authors called their approach the *Siamese* architecture. Incidentally, the problem they addressed with this approach was (on-line) signature verification.

In recent years DNNs have been employed for this task. Chopra et al. made use of the aforementioned *Siamese* architecture for the task of face verification [11]. However, they had the entire feature extraction and embedding performed by a DNN. This approach, where a deep neural network (DNN) is trained to learn the embedding function for the similarity metric, is termed *deep metric learning*.

3.4.2 Improvements of Similarity Metrics

Orthogonal to the use of deep learning, advances have since then been made concerning the question of how best learn the embedding function.

Schultz and Joachims proposed employing *relative* comparisons for this purpose [66]. For this, they embedded three rather than two samples with the same model, two of which were of the same class, one of another. Instead of comparing the absolute positions of the embedded samples, they compared the relative distances of the samples to each other. Thereby, the two samples of the same class should be embedded close to each other, while the third sample (of another class) should be embedded further away.

This approach of embedded *triplets* was finally combined with DNNs by Wang et al. [81], who proposed learning a ranking function for image retrieving based on this.

Shortly after, Hoffer and Ailon showed how this approach could be generalized to work on a broader set of tasks [27], and outperforms the Siamese architecture. In the authors approach, a triplet of samples is forwarded through the same model, similar to [66]. One of the three samples is called the *reference* sample x . Of the remaining two samples, one is from the same class as x , called x^+ , the other is from a different class (x^-). After embedding the samples, their relative Euclidean distances to the reference are compared, this means

$$\|f(x) - f(x^+)\|_2, \text{ and } \|f(x) - f(x^-)\|_2. \quad (3.13)$$

As a means of comparison and determining the loss for the model Hoffer and Ailon make use of the softmax function (compare equation 3.5) and the MSE loss. Since we apply the same method, the loss function is further detailed in chapter 6, which discusses our own implementation.

4 Datasets, Software and Hardware Resources

In this chapter we provide an overview of datasets, the software, and the hardware resources used for the development and training of the system proposed in this thesis.

4.1 Datasets

The system we propose is based on a deep learning approach. It is characteristic for deep learning that training the DNN requires a large amount of sample data. We hence employed a variety of off-line signature datasets, which we briefly introduce here.

| <i>Dataset</i> | <i>Authors</i> | <i>Genuine</i> | <i>Forgeries</i> |
|------------------------|----------------|------------------|-------------------|
| MCYT ₁₀₀ | 100 | 2500 (25 p.a.) | 2500 (25 p.a.) |
| GPDSsyntheticSignature | 4000 | 96 000 (24 p.a.) | 120 000 (30 p.a.) |
| SigWiComp2013 Dutch | 54 | 1292 | 639 |
| SigWiComp2013 Japanese | 11 | 462 (42 p.a.) | 396 (36 p.a.) |

Table 4.1: An overview of the datasets we used. Where possible, the table states how many genuines and forgeries per author (p.a.) are included in the dataset.

An overview to all datasets we used is given in [table 4.1](#). The overview contains information about the total number of authors who contributed original signatures, the number of genuine signatures, and the number of forgeries in the dataset. When forgeries per author are stated, *author* always refers to the author of the genuine signature, that the forgery tries to imitate, not to the creator of the forgery.

Some of the datasets originate from past conference challenges. For these datasets it was not always possible to acquire the complete dataset as used in the challenge. The numbers in the tables below indicate the quantities for the datasets we used in this work, rather than the quantities originally specified for the challenges.

4.1.1 MCYT₁₀₀

The *MCYT baseline corpus* [54] is a dataset of biometric features presented in 2003. The dataset provides fingerprints and on-line signature samples of a total number of 330 individuals. For the signatures, both genuine signatures and forgeries are

provided. MCYT was motivated by the observation that both development and evaluation of biometric systems lack large databases of publicly available data, that has been acquired under realistic conditions [54].

In this project, we used a subset of the MCYT corpus called “MCYT100”, or “MCYT-Signature-100”, which includes 100 signatures and no fingerprints. As this project is concerned with off-line signature verification, we ignored the on-line information and used only the static images.

4.1.2 GPDSSyntheticSignature

The *GPDSSyntheticSignature* [21] corpus is the largest of the datasets we employed. The signatures in this dataset are synthetically generated. For this purpose, the authors of the dataset created a system for synthetic signature generation that is inspired by the human neuromotor system.

Due to its synthetic nature, we use this dataset only in combination with other datasets and only for training the system, not for evaluating it.

4.1.3 SigWiComp2013

This dataset originates from the “ICDAR Competition on Signature Verification and Writer Identification for On- and Offline Skilled Forgeries” from 2013 (SigWiComp2013) [44]. The original challenge provided a variety of datasets, as it was not focused on off-line signature verification only, but involved challenges concerning on-line signature verification and writer identification as well.

In this challenge, off-line signature verification was evaluated on two different sets of signatures: one containing Dutch signatures and one containing Japanese signatures. Both sets included a training and an evaluation set; the Dutch training set was split to two sub-sets (“TrainingA” and “TrainingB”) Of these datasets we were able to obtain part of the Dutch training set (“TrainingB”) and the Japanese training set.

4.1.4 MNIST

MNIST is not a dataset of off-line signatures, but a dataset of handwritten digits. It contains samples originally acquired by the US American National Institute of Standards and Technology (NIST), preprocessed in a way that they can easily be employed in a machine learning system [39]. MNIST provides an overall number of 70 000 samples of handwritten digits from 0 to 9, 10 000 of which make up the evaluation set.

As MNIST does not contain samples of signatures, we did not use it in order to train the neural network model. Instead, we used it for debugging purposes. Since the task of classifying digits based on MNIST is comparably easy, the dataset is well suited for the detection of general flaws in the system’s training procedure.

4.2 Software

The system presented in this thesis is entirely implemented in *Python 3*¹. For matrix based computation we made extensive use of the Python package *NumPy*² [76].

The implementation of the neural network training makes use of the deep learning framework *Chainer*³ [74]. Chainer provides functionality commonly used for DNNs, such as implementations and algorithms for the network's layers, gradient update, and loss computation. Internally, Chainer makes use of NVIDIA's⁴ libraries *CUDA*⁵ for GPU computation and *cuDNN*⁶ [10] for fast deep learning primitives on the GPU.

For other machine learning tasks, such as experiments with K-Nearest Neighbors clustering, we made use of the Python package *scikit-learn*⁷ [57]. The Python packages *SciPy*⁸ [34] and *scikit-image*⁹ [77] were used in the context of image processing, including the augmentation of samples. Graph plotting was implemented using *matplotlib*¹⁰ [31], also a Python package, in combination with the *Seaborn*¹¹ [82] library.

Part of our evaluation is based on the log-likelihood-ratio cost (C_{llr}). The C_{llr} is computed using the *FoCal II* toolkit¹² [8].

4.3 Hardware

The machine we used to train the neural network model possesses an Intel® Core™ i7-4790K CPU, 16 GB of main memory, and two NVIDIA GeForce® GTX 980 GPUs with 4 GB of graphics memory each. We always used one of the GPUs at a time to train the model, not both together. The machine runs the Ubuntu¹³ operating system in version 14.04.

¹<https://www.python.org/> (accessed 20.07.2016)

²<http://www.numpy.org/> (accessed 20.07.2016)

³<http://chainer.org> (accessed 20.07.2016)

⁴<https://www.nvidia.com> (accessed 20.07.2016)

⁵<https://www.nvidia.com/cuda> (accessed 20.07.2016)

⁶<https://developer.nvidia.com/cudnn> (accessed 20.07.2016)

⁷<http://scikit-learn.org> (accessed 20.07.2016)

⁸<https://scipy.org.org> (accessed 20.07.2016)

⁹<http://scikit-image.org/> (accessed 20.07.2016)

¹⁰<http://matplotlib.org/> (accessed 20.07.2016)

¹¹<https://stanford.edu/~mwaskom/software/seaborn> (accessed 20.07.2016)

¹²<https://sites.google.com/site/nikobrunner/jfocal>

¹³<http://www.ubuntu.com/> (accessed 24.07.2016)

5 Writer Independent Signature Verification: Signature Embedding

We propose a new approach to writer independent off-line signature verification. We call this approach *Signature Embedding*. In this chapter we present the basic concepts of our system.

5.1 Approach and Intuition

The key concept of our approach lies in the application of a similarity metric to off-line signature verification. The distance function of the similarity metric is learned using deep learning methods.

In order for the system to be writer independent, it should be able to verify signatures of users whose signatures have not been available during the system's training. Given only a set of genuine signatures of one user and a questioned sample, the system should be able to classify the questioned sample as "genuine" or "forged".

This setup is very similar to the task that a forensics expert could be facing; the approach we propose can be compared to how forensics experts approach the problem: In order to decide whether or not a questioned sample is a genuine signature, forensics experts examine both a set of known genuine (reference) samples and the questioned sample. They use the reference samples to extract a number of features characteristic to genuine signatures of the author. Subsequently, they evaluate how much the questioned sample differs from the reference samples with respect to these features.

In the system we propose, this process is reflected in the following way: The system is given a questioned sample and a number of reference samples. By means of a DNN the system generates a feature representation of each sample. This corresponds to the extraction of characteristic features. Such a feature representation can be understood as an embedding of the sample as a point in a high-dimensional Euclidean space. Where the expert would compare the visual difference of the selected features, our system can use the distance of two samples in the Euclidean space as a similarity metric. The system should embed the samples in a way that all genuine samples of a user are close to each other and all forged signatures are further away from them. Note that this requirement concerns only the relative positions of the signatures to each other in the Euclidean space, but not their absolute positions. Figure 5.1 illustrates this requirement for a two-dimensional space. As a result, the Euclidean distance of the questioned sample to the reference samples can be employed as an estimator of their similarity, and hence of the probability

of the questioned sample to be genuine. The name Signature Embedding derives from the pivotal role of the embedding in our approach.

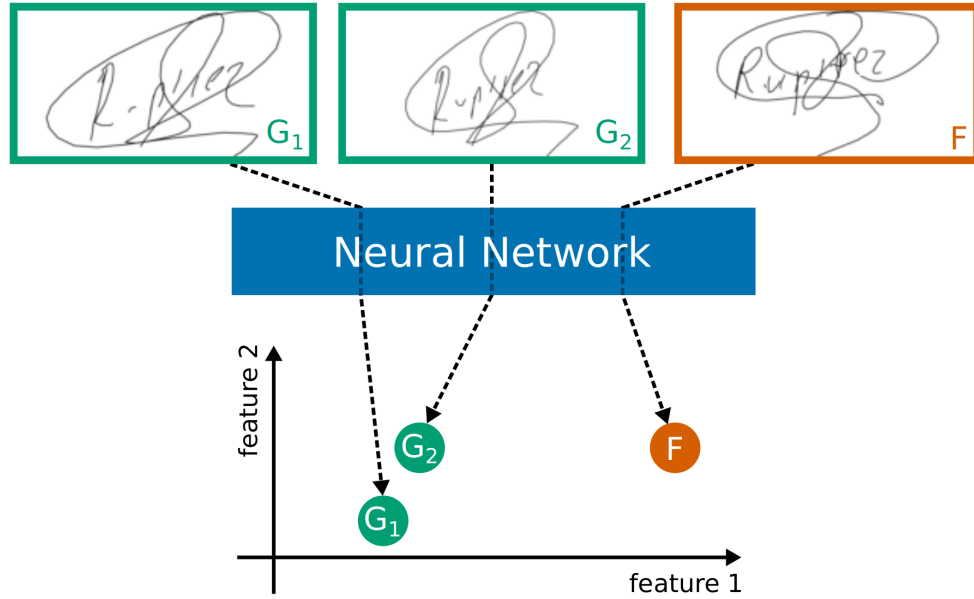


Figure 5.1: The system embeds signatures into Euclidean space by means of the neural network. Genuine signatures G_1 and G_2 are embedded closely together, while a forged signatures F is embedded further away from them.

Summarizing the above, classifying a questioned sample can be reduced to three subtasks:

- embedding it into the Euclidean space,
- calculating its distance to embedded reference samples, and
- evaluating this distance to make a decision of “genuine” or “forged”.

In the remainder of this chapter we will discuss each of these three subtasks.

5.2 Embedding the Signatures

As pointed out above, the embedding of a signature corresponds to a feature representation in an n -dimensional Euclidean space. Accordingly, the process of embedding a signature corresponds to a feature extraction. In deep neural networks the convolutional layers are responsible for feature extraction. Both the convolutional and the fully connected (FC) layers, which follow after them, are trained together to perform a classification task. During this process, the convolutional layers implicitly learn to create appropriate feature representations. If the FC layers were to be removed and a sample was forwarded through the convolutional layers only, the result would be the feature representation.

However, in this process the parameters in the convolutional layers are optimized to provide features exactly for the classes that the FC layers are supposed

to distinguish. This contradicts our goal of creating a writer independent system, as the system should operate on classes (users) it has never seen during training.

Hoffer and Ailon [27] proposed a method to learn feature representations based on distance comparisons, rather than as part of a classification task. We discussed this approach already in the context of similarity metrics in [section 3.4](#). In essence, the authors compare the embeddings of groups of three samples at a time. One such group of three samples is termed *triplet*. They introduce a loss function that minimizes the distance between two of the samples, while maximizing the distance to the other. We employ this loss function in order to minimize the distances between a user's genuine signatures, and maximize the distances between a user's genuine signatures and forgeries. The implementation of this loss function in our system and its effect on the embedding is further detailed in [section 6.4](#).

A noteworthy difference between the Signature Embedding system and the approach described above arises from the fact that our system is concerned with forgeries, rather than just different users. Hoffer and Ailon consider n classes that they want to distinguish and hence cluster in the Euclidean space. In contrast, in Signature Embedding we have n classes for users *plus* their respective forgeries. However, forgeries are not separate classes of their own. The spacial separation of forgeries is no requirement in our system. The only requirement is embedding one user's genuines close together in one cluster and the same user's forgeries somewhere far away from that cluster. In consequence, different users' forgeries are not separated from each other, nor are one user's forgeries separated from another user's genuines. [Figure 5.2](#) illustrates a possible resulting embedding. This figure was generated from actual data embedded by our system. For illustration purposes the dimensionality of the embedding was reduced to two dimensions by means of a *PCA*. Further details about the PCA visualization can be found in [section 7.2](#). Note that the figure also shows that the system is able to cluster the genuine samples appropriately by user. This means for the set of users it was trained on, our system can also perform a user identification. For the scenario of writer independent signature verification, however, this is not of further interest.

5.3 Calculating the Distance

After embedding the signatures, they are represented as points in an n -dimensional Euclidean space. The Euclidean distance of two of points calculates as

$$d(p, q) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}.$$

If more than one reference signatures is available, embedding of the questioned signature is compared to the centroid of the embedded reference signatures. Note that we use the squared Euclidean distance in practice, since distances are only compared to each other. This saves the calculation of the square root.

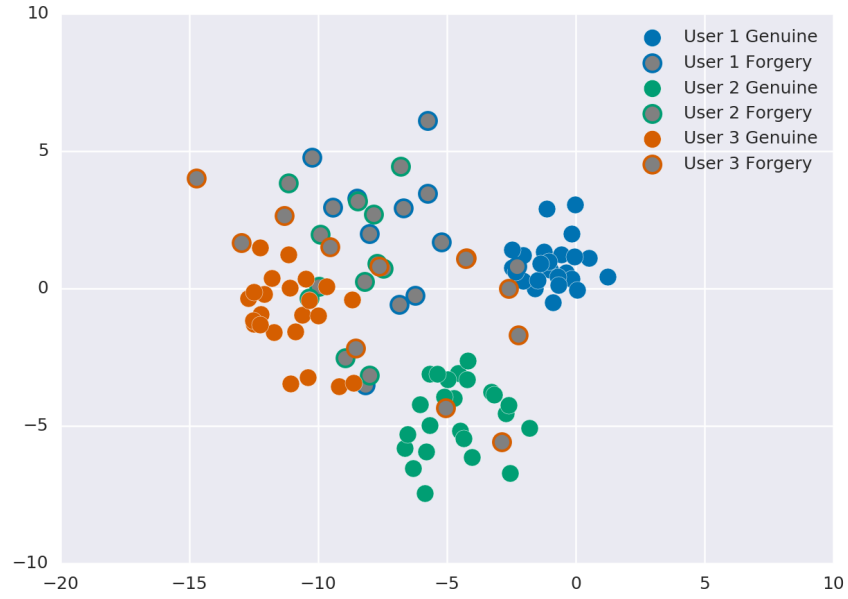


Figure 5.2: The embedding creates clusters that separate genuine signatures of users 1, 2, and 3 from each other, and each user's genuine signatures from the same user's forgeries. Overlaps occur both among different users' forgeries as well as among one user's forgeries and another user's genuine signatures.

5.4 Making a Decision

After signatures have been embedded by the neural network and the distance of the questioned signature to its reference signature has been calculated, the final step is the classification decision based on that distance. In recent years, notable signature verification challenges, such as the ICDAR SigWiComp [44], have adopted the distinction of *hard decisions* and *soft decisions*, based on evaluation practices used for speaker recognition systems [7]. A hard decision detector describes a system that responds to a questioned signature with an absolute answer of "genuine" or "forged". In contrast, a soft decision detector responds with relative value that expresses the system's confidence that the questioned signature is genuine. For real world applications, especially in a legal forensics context, the latter behavior can be a very desirable property.

We will examine both types of decision making in the following.

5.4.1 Hard Decisions: Thresholding

The naïve approach to make the classification decision given a distance is setting a threshold θ . If the distance exceeds θ , then the questioned signature is too far away from the reference signature and hence classified as "forged". Accordingly,

questioned signatures within a distance of θ from the reference signature are classified “genuine”. Choosing θ is the obvious challenge here.

Since the distance is just an absolute number, it is impossible to choose a threshold given one distance output alone. A comparison to distances produced by labeled samples is required. Distances produced by the samples in the training datasets can be used for this purpose.

If the classifier was perfect, it would be possible to deduce a clear threshold, that unambiguously separates genuines from forgeries. To the best of our knowledge, a perfect signature verification classifier does not exist. Hence, θ determines a trade-off between the number of forgeries mistakenly accepted as genuines and the number of genuines mistakenly rejected as forgeries. This trade-off is depicted in figure 5.3.

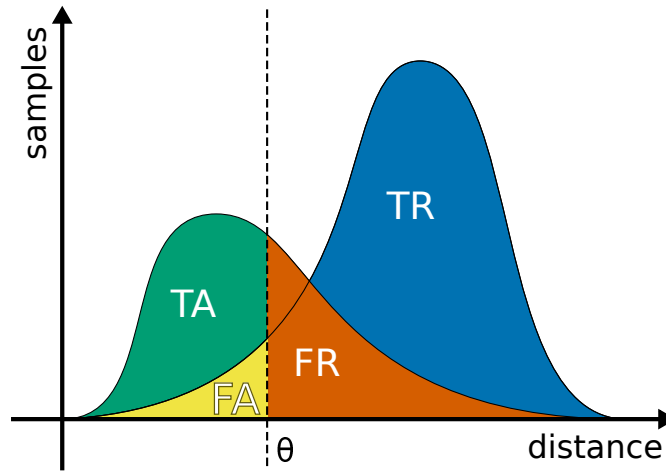


Figure 5.3: The threshold θ determines the trade-off between false accepts (FA) and false reject (FR). In this exemplary plot, increasing θ would increase the number of true accepts, but at the same time increase the number of false accepts.

The decision about this trade-off depends on the application of the system, which determines the severity (*cost*) of each kind of false decision. In some scenarios, false accepts cannot be tolerated. Examples for such scenarios are finance and law forensics, where a person who is mistakenly identified with a signature could face criminal conviction. On the other extreme, there are scenarios in consumer electronics, where usability is very important and users should not be bothered to re-enter their signature too often. Thus, a higher false accept rate would be tolerated for a lower false reject rate.

Systems where the decision about this trade-off has to be made during the design of the system are *application dependent*.

5.4.2 Soft Decisions: Likelihood Ratios

Alternatively to hard decisions, the system could respond with a relative value as a measure of confidence that the questioned signature is genuine. Van Leeuwen

and Brümmer [78] describe how the *log-likelihood-ratio* can be employed as such a relative value. In the original terminology, the log-likelihood-ratio is defined as

$$\mathcal{L}(s) = \log \frac{P(s|\text{target trial})}{P(s|\text{non-target trial})}.$$

In the terminology of signature verification, the log-likelihood-ratio thus describes the logarithm of the ratio between the two likelihoods that a given *score* s is observed for a genuine signature (target trial) and the same score is observed for a forged signature (non-target trial). The scores are derived from the distances that the system computes. The two likelihoods can be computed by observing the distances the system yields on labeled data. The relation of distances to scores and the respective likelihoods is further discussed in [section 7.2](#).

As Van Leeuwen and Brümmer point out, $\mathcal{L}(s)$ can be interpreted as “expressing the degree of support that the raw score s gives to one or the other hypothesis”[78]. In this case, the hypotheses are “questioned sample is genuine” and “questioned sample is a forgery”. $\mathcal{L}(s)$ behaves in such a way that it is close to zero if the support for both hypotheses is about equal, in other words, if the classifier is uncertain. The stronger the support for the “genuine”-hypothesis, the more the fraction grows beyond 1. In consequence, the logarithm of the fraction grows ever larger than zero. In contrast, as the support for the “forgery”-hypothesis grows stronger, $\mathcal{L}(s)$ becomes negative and of larger absolute value.

5.4.3 Application Independent Systems

In addition to the desired property of soft decision making, the log-likelihood-ratio approach allows for *application independent* systems. Note how no threshold had to be chosen in order to obtain the log-likelihood-ratio. We merely observed the distances the system produces with respect to the input data. As a result, the trade-off between false accept and false reject is postponed to not be part of the system design anymore.

This motivates a system design where the decision about trade-off does not have to be taken at design-time, but rather at application-time. Van Leeuwen and Brümmer [78] propose a decision rule that takes these considerations into account:

$$\text{decision}(s, \theta) = \begin{cases} \text{accept} & \text{if } \mathcal{L}(s) \geq -\theta \\ \text{reject} & \text{if } \mathcal{L}(s) < -\theta \end{cases}$$

where

$$\theta = \log \left(\frac{P_{gen}}{1 - P_{gen}} \frac{C_{FR}}{C_{FA}} \right).$$

All application dependent parameters have been moved into the decision threshold θ , which is a function of the prior probability that a genuine signature is observed in the application, P_{gen} , and the costs for false rejects and false accepts, C_{FR} and C_{FA} , respectively. Thus, θ can easily be calibrated per application.

6 Implementation

The core component of the system we propose in this thesis is the neural network model that is responsible for the embedding. In this chapter we examine the design of the neural network and provide details about the implementation of the model's training procedure. This involves four high level tasks:

- designing a neural network layout,
- augmenting and preprocessing the input samples,
- pretraining a model for initialization, and
- training the model.

We discuss each of these in the following sections.

In addition to training the model that performs the embedding, employing this model to obtain actual verification results is of course important as well. However, as pointed out in [section 5.3](#), this amounts to calculating the Euclidean distances between embedded samples and interpreting them to draw an appropriate conclusion.

6.1 Neural Network Design

The underlying neural network layout is crucial in order for the model to be able to fulfill its task. Designing a DNN from scratch is a very difficult exercise, as an in-depth understanding of the inner workings of DNNs has yet to be developed by the deep learning community. The layout of state-of-the-art DNNs usually relies on empirical findings regarding which components work well together and best practices derived from such findings. As a result, the common approach to create a neural network layout for a certain task is to start with an existing layout that has been found to work well in literature. This layout can then be adapted to fit the individual needs, or sometimes used as is. Adapting the layout usually involves scaling it up or down, which in turn relies on best practice methods [71]. Depending on the task, it can also be required to add or remove specific types of layers in order to achieve the desired behavior. Other factors, such as the available hardware resources, take influence on the design process as well: State-of-the-art DNNs usually need to be trained on either a very large CPU-cluster [14] or on one or more GPUs. As the former is not always available, the size of the DNN is often practically limited by the number of parameters that can be held in GPU memory.

Specific to the task, the size of the DNN is determined by the *bias-variance tradeoff* [22], imposing underfitting or overfitting on the resulting model. The bias

| <i>layer type</i> | <i>filter size, stride, padding</i> | <i>output size ($dim \times w \times h$) or number of neurons ($in \rightarrow out$)</i> |
|---------------------|---|--|
| <i>(input data)</i> | <i>(does not apply)</i> | $1 \times 192 \times 96$ |
| convolution | 11, 3, 1 | $96 \times 62 \times 30$ |
| convolution | 3, 1, 1 | $96 \times 62 \times 30$ |
| max pooling | 2, 2, 1 | $96 \times 32 \times 16$ |
| convolution | 5, 1, 1 | $128 \times 30 \times 14$ |
| convolution | 3, 1, 1 | $128 \times 30 \times 14$ |
| max pooling | 2, 2, 1 | $128 \times 16 \times 8$ |
| convolution | 3, 1, 1 | $256 \times 16 \times 8$ |
| convolution | 3, 1, 1 | $256 \times 16 \times 8$ |
| convolution | 3, 1, 1 | $256 \times 16 \times 8$ |
| max pooling | 2, 2, 0 | $256 \times 8 \times 4$ |
| convolution | 3, 1, 1 | $512 \times 8 \times 4$ |
| convolution | 3, 1, 1 | $512 \times 8 \times 4$ |
| convolution | 3, 1, 1 | $512 \times 8 \times 4$ |
| max pooling | 2, 2, 0 | $512 \times 4 \times 2$ |
| fully connected | <i>(does not apply)</i> | $4096 \rightarrow 1024$ |
| fully connected | <i>(does not apply)</i> | $1024 \rightarrow 128$ |

Table 6.1: Layout of the deep neural network

characterizes the prediction error that results from the model’s limited ability to learn relevant relations of features in the training data. Choosing a too small network layout leads to a high bias and thus underfitting. Variance in contrast refers to the error that results from the model responding overly sensitive to variation in the training data. Choosing the network layout too large can result in a low variance and thus in modeling exactly the training data (overfitting).

It is also noteworthy that the process of properly adjusting the layout is an empirical one. The layout of the network interdepends and interacts with other design decisions of the system, especially the loss calculation. It hence needs to be tested in a complete process setup. Nonetheless, it proved a valuable approach to prototype the network layout, as well as the design of the full system, with a simpler task. Specifically, we made use of the *MNIST* dataset for handwritten digits, training the system to distinguish the digits from 0 to 9.

Table 6.1 shows the layout of the DNN we used in our system. This layout is based on the “VGG-16” Network [68], which was proposed by the Visual Geometry Group of the Department of Engineering at the University of Oxford. The original layout has an aggregate number of 16 convolutional and FC layers. In our network, one stack of three convolutions and a pooling operation just before the FC layers has been removed. In addition, output dimensions and filter sizes have been adjusted in the way that proved to work best for our requirements in our

experiments. The output widths and heights have changed as well, as a result of altered input and filter sizes.

6.2 Augmentation and Preprocessing

Preparing the data for being fed into the DNN involves two steps: augmentation and preprocessing.

Augmentation describes the process of extending the size of the dataset used for training. This is a common practice in many application areas of deep learning, such as image recognition (e.g. [38, 67]). Especially in the area of signature verification this step is of great importance, since the lack of labeled training data poses a major challenge.

Preprocessing means preparing the (augmented) data to fit the input format expected by the neural network, for example with regards to the number of color channels, width, and height.

In our implementation, both of these preparation steps are combined in a single process that is applied to each image in the dataset. First, the sample is normalized and cropped in order to remove white areas in the image around the actual signature. Subsequently, we apply a rotation and a perspective distortion. These operations emulate different camera perspectives on the same signature. To each sample, three different rotations and seven different perspective distortions are applied. This gives us a total of 21 augmented samples per base sample in the dataset. Note that the sample should not be distorted too heavily. Especially if the sample represents a genuine signature, a too heavy distortion could make it impossible to recognize. In fact, some authors even use distortion effects in order to emulate forgeries from genuine signatures [29]: augmented genuine samples are generated by applying a weak distortion, while forgeries are created by applying a heavy distortion. The dataset we use, however, already supplies forgeries.

For DNNs there are often very few preprocessing steps needed. In classical machine learning the processing steps before the data can be provided to the neural network include the feature extraction as well. This step is omitted in deep learning. Thus, the only preprocessing required in our system is ensuring a single color channel (grayscale) and resizing the image to the expected size of 192×96 pixels.

All augmentation and preprocessing operations are implemented with the Python libraries NumPy [76], SciPy [34], and scikit-image [77].

Augmentation is done on samples that will be used for training only. For the evaluation we use the unaugmented samples as provided by the dataset and only preprocess them to suit our network's requirements.

6.3 Pretraining

Even though our network is a reduced version of the original VGG-16, it is still quite deep, meaning it has comparably many layers. Training a model for such a

deep network to fulfill a complex task “from scratch”—i.e. starting from randomly initialized parameters—can take very long. It is thus common practice to base the model on an existing, *pretrained* model with the same network layout, but trained for a different, possibly easier task.

6.3.1 Writer Dependent Signature Verification

In fact, the task for the pretrained model does not even need to be closely related to the task the actual model is supposed to fulfill. For a deep network like VGG-16, any model whose convolutional layers are trained to process images provide a better starting point than randomly initialized parameters.

In our scenario, however, a simpler but closely related problem is at hand: writer dependent signature verification on random forgeries. In writer dependent signature verification, the task is to distinguish genuine signature and forgeries for a given set of authors. We further restrict the task to random forgeries. In other words, the DNN is only trained to distinguish the signatures of a fixed set of authors from each other. The model can hence be trained to extract features relevant to exactly these authors’ signatures, which makes the task considerably easier.

6.3.2 Writer Dependent Training

In order to allow for the pretrained model’s parameters to be copied to the actual model, the network layout for pretraining is very similar to the layout described in [section 6.1](#). The only difference is the number of outputs in the final FC layer. While this number indicates the number of embedding dimensions in our main model, it has to match the number of authors in the pretraining. Its output is thus supposed to indicate which of the given authors the signatures belongs to.

The model is trained to fulfill the task in the usual supervised fashion: The DNN is provided with an input sample and the correct label (the author who created the input signature). After the input sample has been forwarded through the network, the loss is computed as cross-entropy loss on the network output and the given label.

6.3.3 Using the Pretrained Model

The convolutional layers in the pretrained model will have adjusted their parameters to extract features relevant to signatures, although just for a limited set of authors. Their parameters hence provide good initialization values for the actual model’s training. The FC layers in the pretrained model will have learned to distinguish the given authors based on the features provided by the convolutional layers. Thus, they are of no use for the writer independent training.

We copy only the parameters of the convolutional layers the new model. The FC layers are replaced entirely. During the actual training of the model the weights will be adjusted further.

The pretraining was implemented using the Chainer [74] framework for deep learning, just like the software for training the main model.

6.4 Training the Model

In general, the training process corresponds to the usual steps to train a deep neural network model:

1. Sampling a *batch* of input data
2. Forwarding it through the DNN and computing the loss
3. Adjusting the model's parameters

This process is repeated until the results seem satisfactory. One such repetition is called *iteration*. In addition, a validation is run after each iteration. The validation serves as an estimation whether the ongoing training improves the results or the training has stagnated.

6.4.1 Batch Sampling

As explained in section 3.1.5, DNNs are trained batch-wise. Batches are forwarded one after another through the DNN. Each batch serves as an estimation of how the parameters of the model are best updated.

Our system implements Hoffer and Ailon's [27] triplet-based loss function. Thus, sample batches in our system are composed of triplets: groups of three samples including two genuine and one forged signature. Following [65], who use a similar approach, we call the two genuines *anchor* and *positive* and the forgery *negative*. The term anchor results from the fact that one of the genuine samples is used for comparison with the two other samples.

Each batch is composed of b triplets, where b is the batch size. The model update process requires each batch of triplets T to be split into sub-batches of anchors A , positives P and negatives N , each of batch size b . The sub-batches are further required to be arranged in a way that the samples from triplet T_i are found at A_i , P_i and N_i , respectively.

The choice of b is not hardcoded in the implementation, but chosen at the start of the training. As mentioned before, each batch represents an approximation that should allow for an estimation of how to adjust the model's parameters. Consequently, a larger batch size is generally preferable, as a too small batch size can result in a bad approximation. However, the batch size is limited for practical reasons: all samples in a batch are processed by the network simultaneously. As a result, they all need to fit into GPU memory. In addition, if the batch allows for an approximation of the desired update to optimize the parameters, a batch

size larger than necessary is wasted effort in terms of computation time. In our experiments we found batch sizes around $100 < b < 200$ to be feasible.

In our system the component responsible for loading the sample data from the hard disk and arranging them into batches decoupled as its own modular component. This *data loader* component works on the basis of lookup table that indicate the storage paths to the samples, as well as their labels—in other words, which user the signature belongs to and whether it is genuine or forged.

The data loader has the additional concept of easy and hard triplets. Easy triplets refer to triplets whose negative is a random forgery. Here our system uses other users' genuine or forged signatures as random forgeries. In a hard triplet, in contrast, the negative is a skilled forgery for the same user of which anchor and positive were sampled. Of course the recognition of skilled forgeries is a major requirement for our system. Thus, our batches consist mostly of hard triplets. The ratio of easy and hard triplets within a batch can, however, be configured at the beginning of the training.

Training a DNN can require processing a very large number of batches. It is thus important to ensure that loading and assembling batches is quick. In order to not interfere with the remaining training process, we implemented separate data loading threads. Hence, while one batch is being forwarded and back-propagated through the DNN, the subsequent batch can already be loaded into GPU memory.

6.4.2 Forward Pass and Parameter Update

Recall the task the model is supposed to solve: It should embed anchors, positives, and negatives into Euclidean space, in a way that one user's genuine signatures are embedded close to each other, while forgeries imitating this user are embedded far away from them.

Considering the neural network design presented above, this means the convolutional layers should learn to extract features relevant to the task of signature verification, while the FC layers “translate” these features to a 128-dimensional representation in Euclidean space. Given these representations for anchor, positive, and negative samples, it is possible to compute a loss, that indicates how well the embedding fulfills the task we stated. This loss is aptly termed triplet loss. We implemented this loss function using the basic building blocks provided by the Chainer framework. We were also able to contribute parts of the implementation back to Chainer (see [section 1.2](#)).

Loss Calculation

The complete process of forwarding anchors, positives and negatives through the network and calculating the loss based on the resulting embeddings is depicted in [figure 6.1](#).

In the beginning of this process, three sub-batches of anchors, positives, and negatives have been obtained from the data loader. Each sub-batch is forwarded through the DNN subsequently, resulting in the aforementioned 128-dimensional

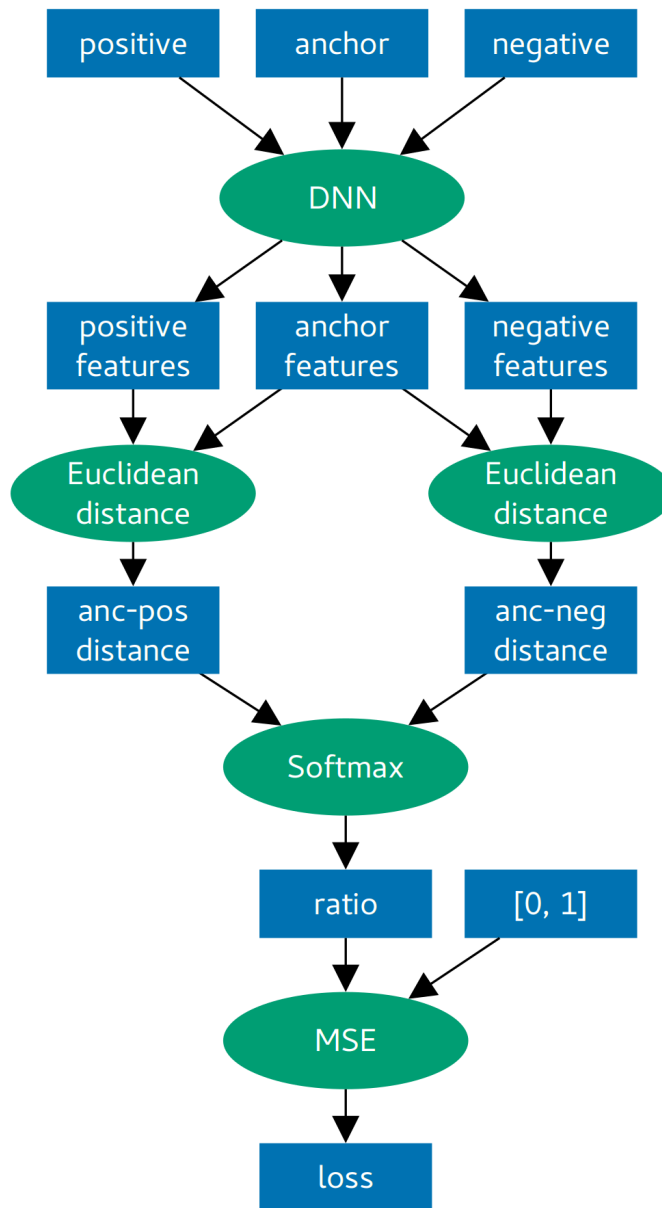


Figure 6.1: The loss is computed based on the *softmax ratio* between the *anchor-positive* distances and the *anchor-negative distances*. For that, anchor, and negative samples are embedded by the DNN, and the distance of the embedded anchor to each of the two other embedded samples is calculated. The final loss is determined by the *mean squared error* of the calculated softmax ratio to the desired ratio $[0, 1]$.

feature representations. The goal is to minimize the distances between anchors and positives, while maximizing the distances between anchors and negatives. Note that the comparison of these distances needs to be done sample-wise rather than sub-batch-wise, since only samples at the same index in each sub-batch form valid triplets. The anchor-positive and anchor-negative distances can be trivially obtained by computing the Euclidean distances between the embedded representations.

In order to evaluate these distances we need a means of comparing them. For this purpose, the *softmax* function is employed. Softmax is defined as

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}, \text{ for } j = 1, \dots, K.$$

This function takes a K -dimensional input vector \mathbf{z} and returns a K -dimensional vector of values in the range 0 to 1, that sum to 1. Softmax is commonly used in machine learning as a part of *cross-entropy loss*, where it serves as a ratio measure between classifier outputs (see [section 3.1](#) for more details). In the case of triplet loss, softmax is employed as a ratio measure between anchor-positive distances and anchor-negative distances. Here, \mathbf{z} is one pair of distances, as also the distance comparison has to be done per triplet, and K is 2. In the beginning of the training the output of the softmax function will be, on average, close to the vector $[0.5, 0.5]$, as the distances will be random, but about equal on average. As the training proceeds, the value of

$$\text{softmax}([\text{anchor-positive distance}, \text{anchor-negative distance}])$$

should move towards $[0.0, 1.0]$, as anchor-negative distance should become much larger than anchor-positive distance. Figure 6.2 illustrates how the embedding of the triplets changes over time.

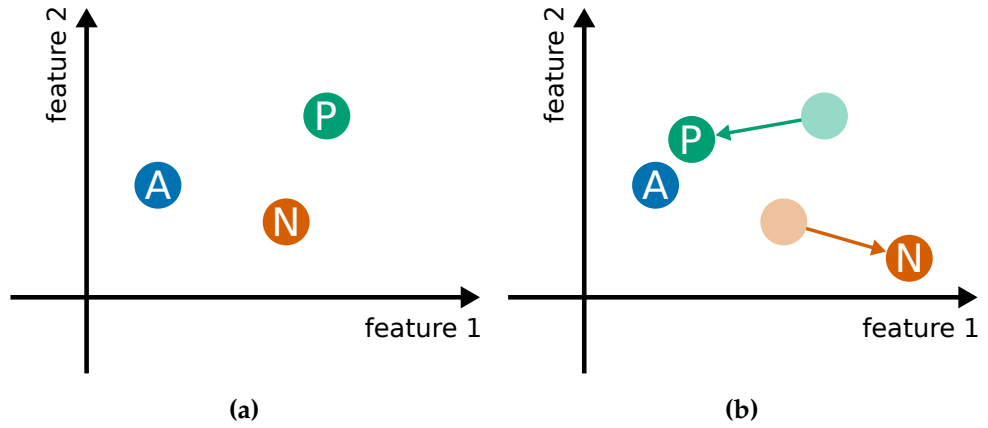


Figure 6.2: Initially, anchors, positives, and negatives are embedded randomly in the Euclidean space (a). As the training of the DNN proceeds, anchors and positives are embedded more closely to each other, while negatives are embedded further away from the anchor (b).

In order to compute a loss from the softmax ratio it is compared to the target vector $[0.0, 1.0]$ using *mean squared error*.

Finally, the loss is back-propagated through the DNN in order to adjust its parameters. The back-propagation mechanism is provided by the Chainer framework.

6.4.3 Validation

After each iteration of forwarding and back-propagating a batch through the model, the updated model is validated. Therefore, a batch of test samples—samples that are not part of the training data—is forwarded through the model. The resulting loss and embeddings can be used for evaluating the current model’s performance. The loss is, however, not back-propagated and the model parameters are not adjusted, as the model is not supposed to learn from the test samples.

The loss already provides an indicator of how well the training proceeds, as it should reduce over the course of the training. However, in absolute numbers the loss is not suitable for drawing conclusions about the model’s performance. Thus, the *accuracy* is commonly employed as an additional metric. The accuracy describes the fraction of correctly classified samples among all samples in the test batch. This fraction can easily be obtained in a system where individual labeled samples are to be classified. Writer dependent signature verification typically is such a scenario. Here, the algorithm can simply verify whether or not the classifier assigned the correct label to a sample.

In contrast, the DNN in our writer independent system produces embedded samples rather than labeled samples. Though we know which samples are genuine and which are forged, the system is required to make a decision (section 5.4), which cannot trivially be done during the training process. It is hence called for other validation measures, that are more suitable for the triplet loss based training.

The loss itself can still provide a good indicator. As an approximation to accuracy we examine the anchor-positive and anchor-negative distances that are produced during loss calculation. In order to emulate a “fraction of correctly classified” on triplets, we count the number of triplets where the anchor-negative distance is larger than the anchor-positive distance. This provides a reasonable indicator, as a higher share of such triplets indeed suggests a better embedding. However, no conclusions can be derived about whether a general threshold, that separates genuines and forgeries for all users, can be found.

Further measures we considered in order to validate the ongoing training are the mean difference and the “worst” difference between anchor-positive and anchor-negative distances. As the perfect classifier would create embeddings such that

$$\exists \theta : \forall u \in Users : \forall anchor, positive \in u_{genuines}, negative \in u_{forgeries} : \\ dist(anchor, positive) < \theta < dist(anchor, negative),$$

where θ is a distinct threshold between all users’ genuine signatures and all users’ forgeries. For validation purposes, the anchor-positive and anchor-negative

distances are hence compared on the complete batch, rather than triplet-wise. The mean difference is thus computed as

$$diff_{mean} = mean(dist(anchor, negative)) - mean(dist(anchor, positive)),$$

where $mean(dist(anchor, negative))$ and $mean(dist(anchor, positive))$ are the respective means across the whole batch. The better the embedding, the larger this number should be.

The worst difference is computed as

$$diff_{worst} = min(dist(anchor, negative)) - max(dist(anchor, positive)).$$

The resulting value indicates the margin for the aforementioned threshold θ . A larger $diff_{worst}$ is obviously better, but since the system cannot guarantee a perfect classification this number is usually negative.

The worst difference was ultimately not employed as a validation metric. It is able to provide an insightful metric to the system's performance in some cases, but since it considers only the worst pair of distances in the batch, it is very susceptible to outliers. Consequently, a single poorly embedded sample can result in a misleadingly poor value in this metric.

In addition to these indicators about the quality of the embeddings created by the model, we found the average time required for a complete forward pass and back-propagation useful in order to monitor the training process for irregularities.

6.4.4 Epochs

Multiple iterations are grouped in *epochs*. In our system one epoch is defined as one iteration per user in the training database. After each epoch it may be required to carry out administrative tasks. These include creating snapshots of the model, that can be used to continue a suspended training process and to use the model for the actual verification task, as well as adjusting hyperparameters, such as the learning rate. The interval in which these tasks are executed is configured at the beginning of the training.

7 Evaluation

In the following chapter we evaluate the Signature Embedding system. We first explain the setup of our experiments and introduce the evaluation metrics we employ. Subsequently, we present the results of the evaluation and compare these results to other systems. Finally, we discuss the implications of a number of design decisions in the training and evaluation process.

7.1 Setup

In this section we describe how the training of the final model, on which the evaluation in this chapter is based, was set up and how it has been employed to obtain the results.

7.1.1 Data

In order to train the model, we used the datasets “MCYT 100” [54], “GPDSSyntheticSignature” [21], and a subset of the “Dutch Offline Signatures” training set from the ICDAR SigWiComp2013 challenge [44]. All datasets have been introduced in greater detail in [section 4.1](#). Samples for 20 authors from the “Dutch Offline Signatures” set have been reserved for evaluation purposes. Neither these authors’ genuine signatures nor any forgeries supposed to resemble these authors’ signatures have been used to train the model. In addition, we used “Japanese Offline Signatures”, also from the ICDAR SigWiComp2013 challenge, for evaluation. Note that no Japanese signatures have been used for training at all.

In order to provide an intuition about the data we evaluated our system with, [table 7.1](#) shows a number of arbitrarily selected genuine and forged signatures from both the “Dutch Offline Signatures” and the “Japanese Offline Signatures” dataset.

The data used for training has been augmented as described in [section 6.2](#). The evaluation data has been preprocessed to suit the model’s requirements concerning input width, height, and depth, but not augmented.

7.1.2 Training of the Neural Network Model

The design of the neural network is based on the VGG-16 network [68] and is detailed in [section 6.1](#).



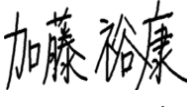





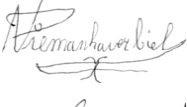

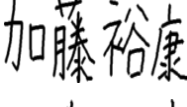
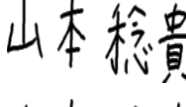
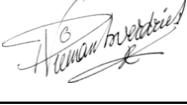

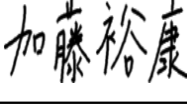
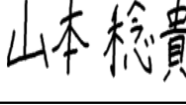
| | Dutch | | Japanese | |
|---------|---|---|--|---|
| | User A | User B | User C | User D |
| Genuine |  |  |  |  |
| |  |  |  |  |
| Forged |  |  |  |  |
| |  |  |  |  |

Table 7.1: Selected samples from the SigWiComp2013 “Dutch Offline Signatures” and the SigWiComp2013 “Japanese Offline Signatures” dataset.

Pretraining

This model is deep enough to make it hard to train it from scratch. Therefore, rather than randomly initializing the parameters of the DNN, we pretrained a writer dependent classifier on “MCYT 100” and used this model’s parameters to initialize the DNN’s parameters. More details about the concepts pretraining can be found in [section 6.3](#). The training of the writer dependent classifier was run until it reached a test accuracy of more than 90 %. Given the comparably simple task of writer dependent classification this took less than 15 minutes.

Configuration and Hyperparameters

Besides the design of the neural network, the hyperparameters used to train the model are of great importance. In the following, we briefly explain the relevant hyperparameters and state how they were configured for the training. Further configuration includes the choice of the gradient update method and the duration of the training, both of which are no hyperparameters in a strict sense, but will also be detailed here.

GRADIENT UPDATE METHOD¹ The gradient update method describes the optimization algorithm that is used to update the model’s parameters based on the computed loss. We used momentum [63], which is a variant of SGD.

LEARNING RATE¹ The learning rate is a parameter to the optimizer. It is a factor with which the gradient update is scaled. We initialized the learning rate with 0.001. The learning rate is commonly decreased over time during the training. We decreased our learning rate every five epochs by a factor of 0.5.

BATCH SIZE¹ The batch size defines how many samples are forwarded through the model in each iteration. We set the batch size to 180 triplets, or 540 samples.

¹A more detailed explanation of these parameters can be found in [section 3.1](#)

HARD TRIPLETS The ratio of hard triplets is a parameter specific to Signature Embedding. It determines the share of triplets that use a skilled forgery as negative, while the rest of the batch uses random forgeries (other users' signatures) as negatives. In our setup the percentage of hard triplets was 90 %.

WEIGHT DECAY Weight decay is a regularization method that lets the model's parameters deteriorate over time. This can help to overcome local minima and avoid overfitting, and thus improve the model's generality. We used a weight decay factor of 0.001.

TRAINING DURATION The model was trained for 55 epochs, until the decrease of the loss stagnated. This corresponds to about 48 hours.

7.1.3 Validation of the Training Process

During the training of the model, we observed the validation metrics described in [section 6.4](#). These metrics we are the loss, the accuracy, and the mean difference of distances. The course of these measures over the time of the training is depicted in [figure 7.1](#).

For both loss and accuracy it becomes apparent that very good values are already reached after about 20 epochs, and only slight improvement can be observed after that. In the final epoch of our training, the average validation loss was 0.059, the average validation accuracy 91.97 %.

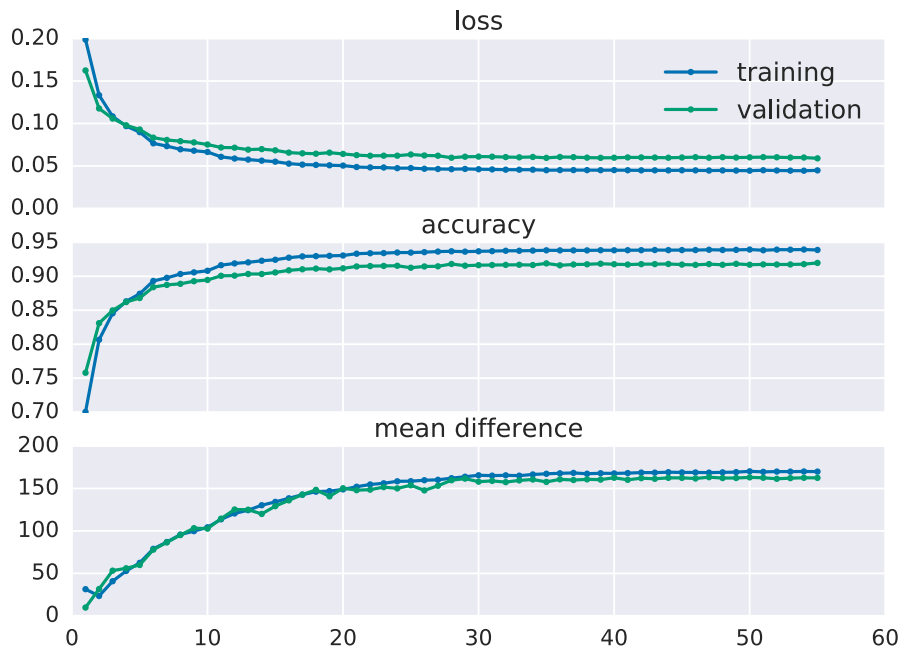


Figure 7.1: Loss, accuracy, and mean difference are continuously improved over the course of the training.

The mean difference is continuously improved as well, although it stagnates later than loss and accuracy. Recall that a large mean difference is desirable, as it indicates how much further from the negative samples are embedded from the anchor, compared to positive samples.

Comparing the performance in all metrics on the training set with the performance on the validation set shows only a slight divergence of the two curves. This shows that overfitting occurs to a small extent, which is to be expected and can hardly be avoided.

7.1.4 Number of Reference Signatures

The performance of the system depends on how many reference signatures can be used. The more reference signatures are available for comparison, the more accurately the system is able to predict whether the questioned signature is genuine or forged.

In order to obtain the distance on which the evaluation is based, both reference signatures and the questioned signature are embedded by the DNN. If only one reference signature is available the distance can directly be calculated as the squared Euclidean distance of the two embedded samples. If multiple reference signatures are available, the centroid of the embedded references is computed and used for distance calculation.

In our evaluation we use 12 reference signatures. This corresponds to the number of reference signatures provided per author in the “Dutch Offline Signatures” dataset.

A writer independent signature verification system should be able to operate with as few reference signatures as possible. In [section 7.5](#) we examine the impact of the number of reference signatures used on the system’s performance.

7.2 Evaluation Metrics

In this section we discuss the evaluation metrics that we employ.

This section is split to three parts. The first and the second part are concerned with the evaluation of the system as an application dependent and as an application independent system. We discussed application (in-) dependent systems in [section 5.4](#) in the context of making hard or soft decisions. The third part is concerned with the possibility of a visual evaluation of the embedding that the system creates.

7.2.1 Application Dependent Evaluation

An application dependent system depends on information specific to the domain in which the system is supposed to be used. Most notably this information includes the cost for a false accept and for a false reject decision. As a result, it is required to choose a threshold that determines a trade-off between these two types of errors.

In this section we discuss evaluation metrics that depend on a threshold that is chosen in beforehand.

Accuracy and F-Measure

Accuracy and F-measure (or F_1 score) are common measures in machine learning classification scenarios. The Accuracy denotes the ratio of correctly classified samples to all samples in the evaluation set. In the case of signature verification the accuracy thus calculates as

$$Accuracy = \frac{|\{\text{accepted genuines}\} \cup |\{\text{rejected forgeries}\}|}{|\{\text{genuines}\} \cup |\{\text{forgeries}\}|}. \quad (7.1)$$

The F-measure is defined as

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}, \quad (7.2)$$

where *precision* denotes the ratio of accepted genuines to all correctly classified samples and *recall* the ratio of accepted genuines to all genuines. The motivation for the F-measure is to balance the precision and the recall score. A high precision could be reached by simply rejecting all samples, as this would avoid false accepts. However, doing so would diminish the recall, as the number of false rejects is considered here. The F-measure thus provides a scoring metric that takes both precision and recall into account.

ROC Curves

Another common metric for machine learning classifiers are *receiver operating characteristic* (ROC) *curves*. A ROC curve is a graphical plot that illustrates an application dependent system's performance in dependence of the chosen threshold. This is implemented by plotting recall against the *false accept rate* (FAR), the ratio of accepted forgeries to all forgeries. In this context, the recall is also referred to as *true accept rate* (TAR).

ROC curves can thus be used to analyze the threshold dependent trade-off between TAR and FAR. Due to the nature of this trade-off, judging over a system's performance based on the ROC curve depends on the application of the system. Both of TAR and FAR lie in the interval of 0.0 to 1.0. A perfect classification system would achieve a TAR of 1.0 and an FAR of 0.0 at a given threshold.

As a general estimation of the system's performance it is sometimes observed *how close* the ROC curve approaches the point (0.0, 1.0). In the evaluation results section below we indicate the points in the ROC curve that were produced by the thresholds that yield the best accuracy and F-measure.

Error Rates

The off-line signature verification community, most notably the ICDAR SigWi-Comp challenges starting from 2011 [40], have adopted two error rates as a performance metric for application dependent systems. One of these error rates is the

aforementioned FAR. The other error rate is the *false reject rate* (FRR), the ratio of rejected genuines to all genuines.

In addition, the accuracy of the system is measured for the threshold that produces the *equal error rate* (EER), that is, the point where the FAR is equal to the FRR. This distills the FAR-FRR trade-off down to a single value, and provides a concise summary of the system's *discrimination* capability.

7.2.2 Application Independent Evaluation

Application independent signature verification systems rely on a classifier that produces *soft decisions*. One way to implement soft decisions is the *log-likelihood-ratio* ($\mathcal{L}(s)$) [7], i.e. the logarithm of the ratio between the two likelihoods that a given *score* s is observed for a genuine signature and the same score is observed for a forged signature. The log-likelihood-ratio, soft decision classifiers, and application independent system were motivated and further detailed in [section 5.4](#).

Since 2011, ICDAR SigWiComp challenges have adopted an application independent metric based on the $\mathcal{L}(s)$: the *log-likelihood-ratio cost function* (C_{llr}) [7]. The C_{llr} provides a metric of both discrimination and *calibration*, the ability to set good decision thresholds [78]. Briefly, the C_{llr} is calculated as the integral of the *expected cost of detection errors* over all possible thresholds. This considers application dependent parameters that determine the expected cost for each type of error (false accept or false reject). These parameters are unknown to the evaluated system. Implementation-wise, C_{llr} is computed based on the $\mathcal{L}(s)$ the system produces for each sample s .

The C_{llr} represents an information loss. The C_{llr} is hence a positive value with $0 \leq C_{llr} \leq \infty$ [8], where a perfect classifier would have a C_{llr} of zero. $C_{llr} = 1$ represents a reference value, that is produced if the system produces the same $\mathcal{L}(s)$ for every sample s , in other words, cannot discriminate between genuine signatures and forgeries at all.

In our evaluation, we use the *FoCal II* [8] toolkit for the computation of C_{llr} . The toolkit additionally performs a calibration of the system. This means the log-likelihood-ratio outputs of the evaluated system are adjusted in order to best fit the error costs encoded in the C_{llr} calculation. Of course the toolkit cannot change the output of the system. It merely scales and translates all log-likelihood-ratios produced by the system with the same parameters. Hence, the calibrated log-likelihood-ratio $\mathcal{L}(s)'$ is calculated as

$$\mathcal{L}(s)' = r \cdot \mathcal{L}(s) + t, \quad (7.3)$$

where the same r and t are applied to all samples' $\mathcal{L}(s)$.

Given these calibrated log-likelihood-ratios, the toolkit computes C_{llr}^{min} , the *optimized* C_{llr} . It is of course legitimate for the signature verification system to already perform a calibration of the log-likelihood-ratios before they are handed to the FoCal II toolkit. Hence, a system that is well configured and thus produces a C_{llr} close to the C_{llr}^{min} found by the toolkit, is very desirable.

In order to obtain *scores* in our system, distances are normalized using the largest distance observed and scaled by a constant factor.

7.2.3 Visualization

Finally we want to consider evaluation methods that are less suitable for providing an accurate comparison, but are able to give an intuition about how well the embedding complies with the similarity metric requirement. For this purpose, the embedding performed by the DNN should be visualized.

The embedded representations of signatures are high-dimensional—in our system 128-dimensional—vectors. In order to visualize this high-dimensional data, a means of dimensionality reduction is required.

For that purpose, a principal component analysis (PCA) [28] can be employed. If the DNN embeds the samples well, each author’s genuine signatures should form clusters in the high-dimensional space, which the author’s forgeries are not part of. This relationship should then be reflected in the PCA. Nonetheless, the PCA entails a loss of fidelity. The clear separation of genuines and forgeries, if it exists in the high-dimensional space, does not necessarily show in the low-dimensional PCA. In the evaluation results section below we show a two-dimensional and a three-dimensional PCA of the embeddings produced by our system.

A more elaborate approach to dimensionality reduction is the *t-SNE* [41]. *t-SNE* is a machine learning based algorithm that outperforms previous visualization approaches on a many datasets [41]. However, since the purpose of the visualizations in this thesis is merely to provide an intuition as to whether or not the embedding works as desired, we employ the simpler PCA method.

7.3 Results

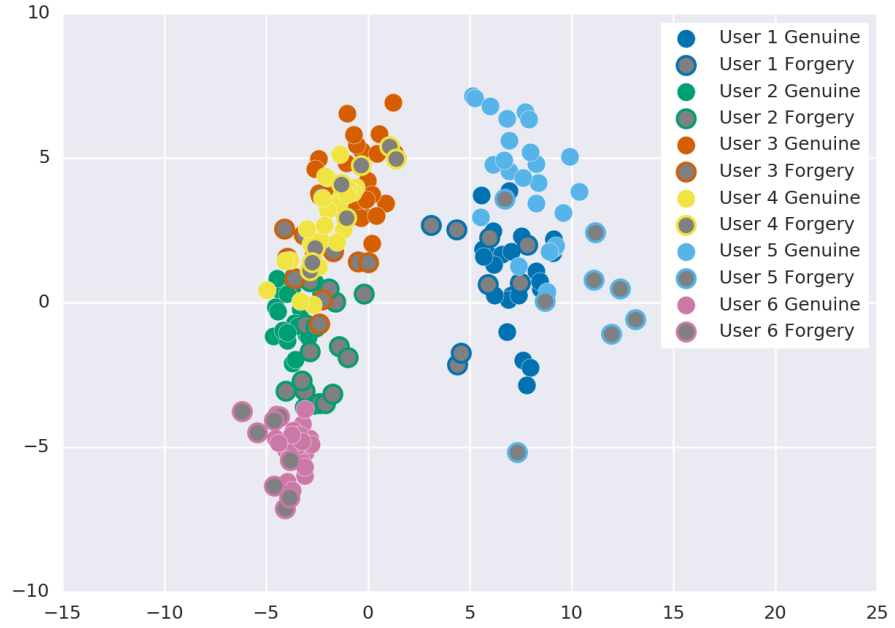
In this section we apply each of the evaluation metrics described above to the system presented in this thesis. As noted above, we used the Dutch and Japanese off-line signature datasets from the ICDAR SigWiComp2013 challenge.

We begin with the PCA, as it provides a good intuition to the embeddings.

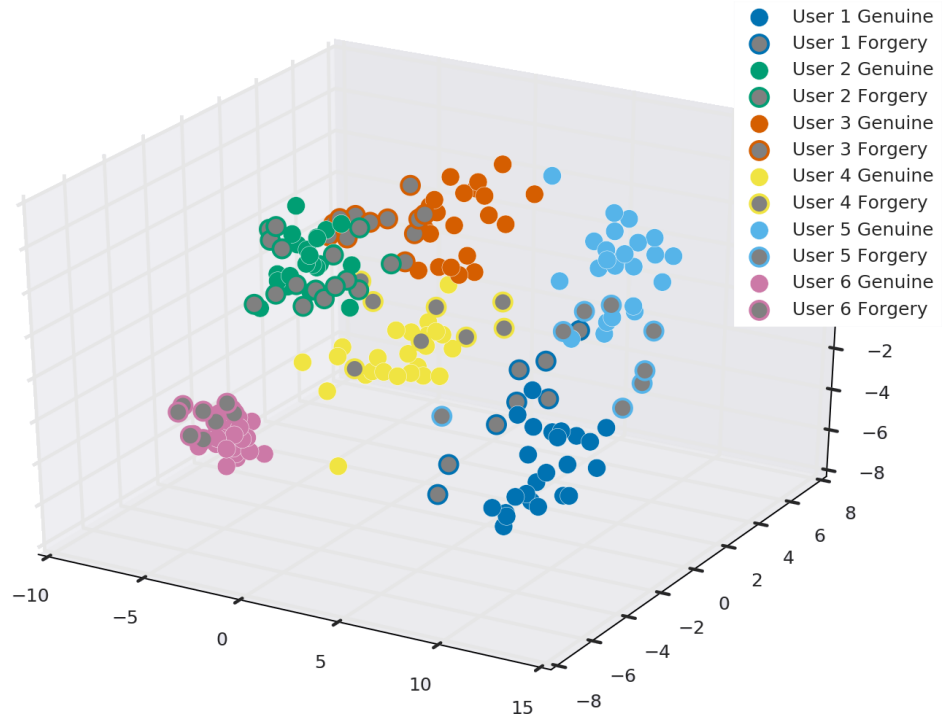
7.3.1 PCA Visualization

Figure 7.2 shows the embeddings produced by our DNN reduced by means of a PCA to (a) a two-dimensional and (b) a three-dimensional space. The figure illustrates that, although both plots are based on the same data, the clustering is more distinct in the higher dimensional space.

The PCAs were generated for a selected subset of users from the Dutch dataset. For these users, the clusters of genuine signatures are overlapping strongly with each other and with the respective forgeries in the two-dimensional visualization. In the three-dimensional illustration the clusters can be better recognized. It becomes apparent that genuine signatures of different users form distinct clusters



(a) two-dimensional representation of the embedding



(b) three-dimensional representation of the embedding

Figure 7.2: PCA visualizations of the embeddings produced by the DNN, reducing the high-dimensional data to the (a) two-dimensional and (b) three-dimensional space.

with almost no overlap. The forgeries for each user do pierce the user's genuine signatures' cluster in a number of cases. From the PCA visualization it cannot be judged whether the reason for this is a shortcoming of the DNN's embedding or the inaccuracy of the low-dimensional visualization².

7.3.2 Distribution of Distances

All evaluation metrics below result from the distances calculated for samples in the respective evaluation set. The application dependent metrics rely on a threshold to be chosen with regard to these distances, the application independent metrics rely on the distribution of distances with respect to the classes of genuine signatures and forgeries. In order to obtain the distance value for any given signature (genuine or forged), we calculate the squared Euclidean distance to the centroid of reference signatures.

Figures 7.3 and 7.4 show normalized histograms of the generated distances for the Dutch and the Japanese dataset, respectively. From both figures it becomes apparent that, in most cases, forgeries are embedded further away from the reference signatures than genuine signatures. The histograms also show, however, that it is not possible to find a distinct threshold between the two groups. As a result, a certain rate of error is inevitable and the choice of the threshold will have to be taken under consideration of the aforementioned trade-off between false accepts and false rejects.

Comparing figure 7.3 and figure 7.4 suggests that the system created better embeddings for the Japanese samples, even though it was never trained with any Japanese signatures.

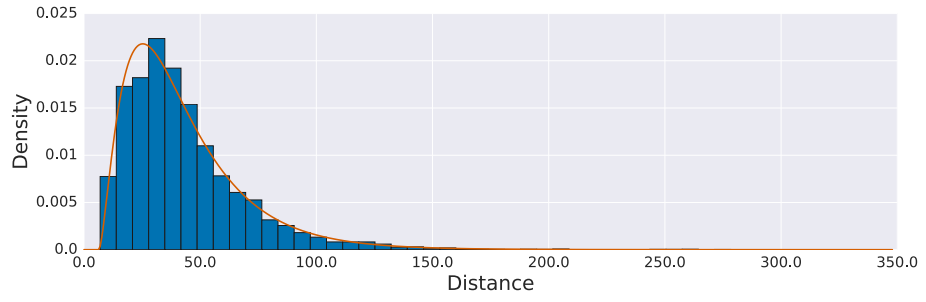
7.3.3 Application Dependent Evaluation Results

Figures 7.5 and 7.6 show the ROC curves for the Dutch and the Japanese dataset, respectively. In the ROC curve diagrams we indicated the threshold values that produced the best accuracy, the best F-measure (F_1), and the EER. For the Japanese dataset these values lie close together. The best accuracy and best F-measure are produced by the same threshold, the EER is produced by a slightly larger threshold. For the Dutch dataset, in contrast, the thresholds producing each of the three points are more divergent.

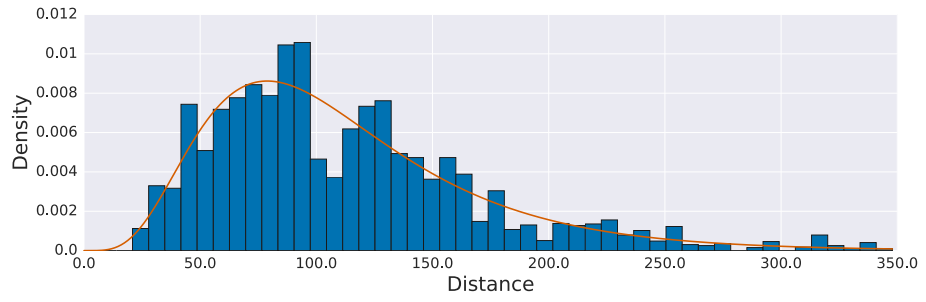
It also becomes apparent that the ROC curve for the Japanese dataset rises more steeply and approaches the top left of the chart more closely. The ROC curves thus clearly indicate that the system indeed performs better on the Japanese evaluation dataset, as the histograms of distances (figure 7.3 and figure 7.4) suggested.

The system's values for the application dependent evaluation metrics introduced above are listed in table 7.2. In this table the values for FRR and FAR refer to those rates that have been considered as the EER.

² Note that the very representation of the three-dimensional visualization as a two-dimensional graphic induces a loss of information as well.

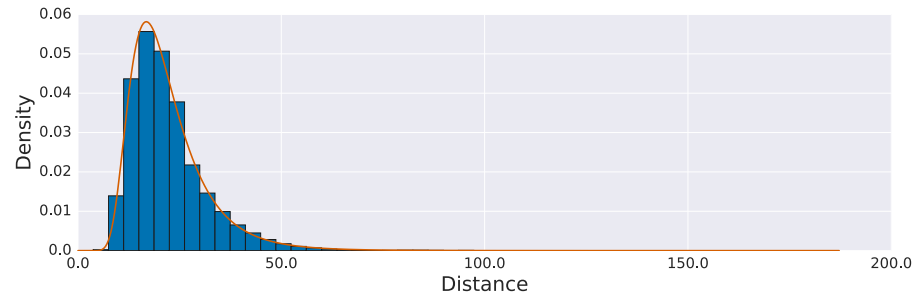


(a) Histogram of genuine signatures' distances

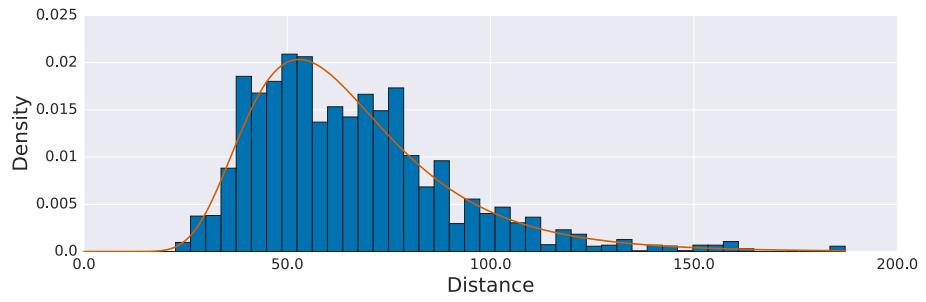


(b) Histogram of forged signatures' distances

Figure 7.3: Normalized histograms of distances from the Dutch evaluation dataset



(a) Histogram of genuine signatures' distances



(b) Histogram of forged signatures' distances

Figure 7.4: Normalized histograms of distances from the Japanese evaluation dataset

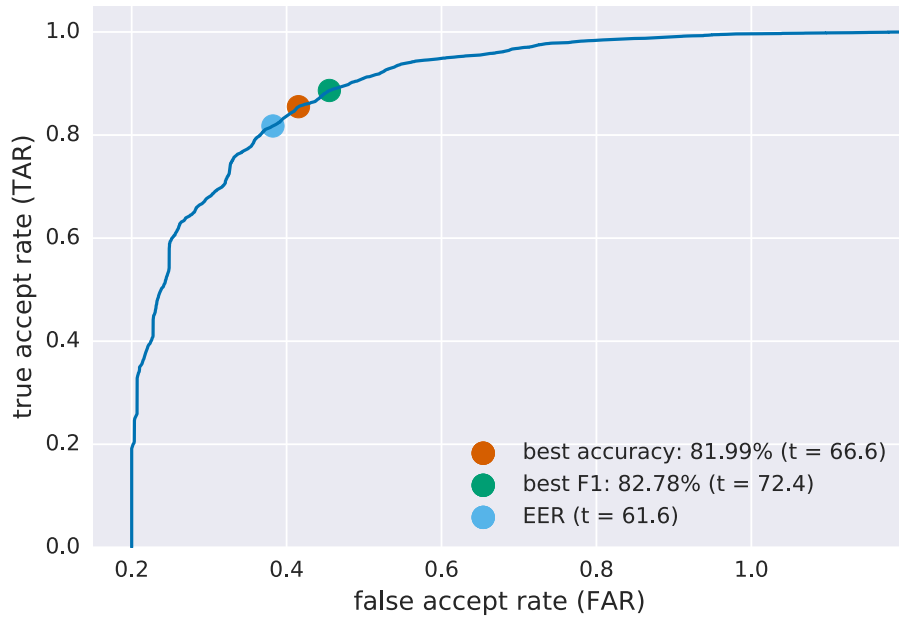


Figure 7.5: ROC curve for the Dutch evaluation dataset

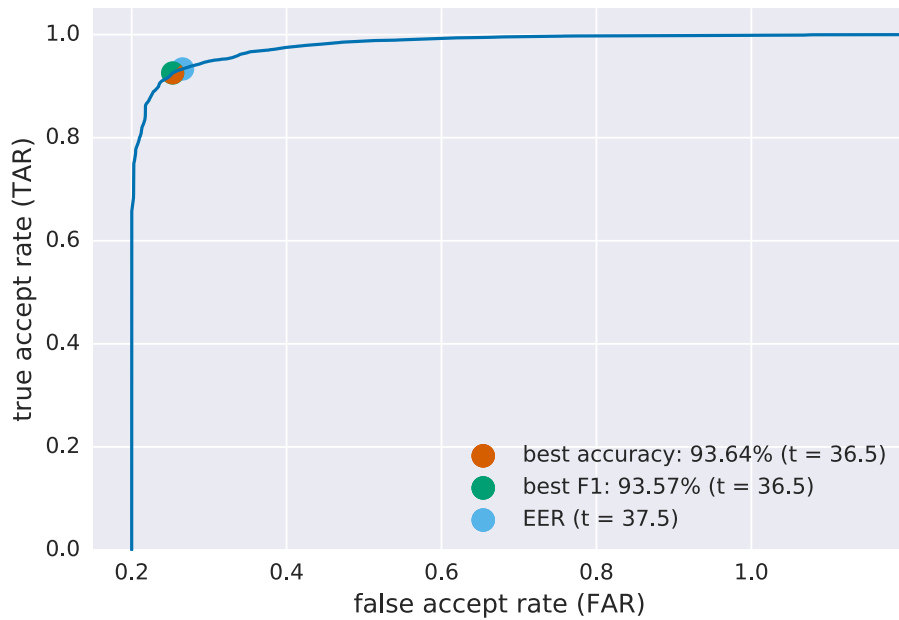


Figure 7.6: ROC curve for the Japanese evaluation dataset

7.3.4 Application Independent Evaluation Results

Table 7.3 shows the results our proposed system produced on the application independent evaluation metrics. The system achieves very good values for these metrics that compare favorably to other approaches, as we will discuss in section 7.4. For both datasets, the C_{llr} is clearly below the reference value of 1.0. Furthermore, the system is reasonable calibrated for both of the datasets, since $\Delta_{C_{llr}}$, the difference of C_{llr} and the calibrated C_{llr}^{min} , is comparably small.

In agreement to the results based on application dependent metrics, Signature Embedding performs substantially better on the dataset of Japanese signatures.

7.4 Discussion and Comparison

The results described above compare very favorable to the results of the ICDAR SigWiComp2013 challenge, which used the same datasets. Tables 7.4 and 7.5 show the Signature Embedding system in comparison with the five best ranked³ competitors in the respective ICDAR SigWiComp2013 challenge [44]. Please note that the evaluation of our system is based on a different subset of the datasets than the results we compare them to. The reason is that the complete datasets were not available to us anymore. We thus, as described above, reserved part of the Dutch training dataset and the complete Japanese training dataset for evaluation purposes and did not use them in order to train our system. Nonetheless this process should provide a very good indication of the system’s performance.

Tables 7.4 and 7.5 show the systems’ values for both the application dependent and the application independent metrics discussed above. In the tables, accuracy, FRR, and FAR refer to the values produced at the equal error rate. Both tables are sorted by the systems’ C_{llr}^{min} values.

According to this metric, Signature Embedding ranks second in the first task of the challenge, as table 7.4 shows. This task was concerned with off-line signature verification on the database of Dutch signatures. It is noteworthy that Signature Embedding scores better than all other systems in all of the other metrics.

The same holds true for the second task, the verification of Japanese off-line signatures. Here our system ranks best in all of the metrics (table 7.5).

The question remains why Signature Embedding performs so much better on the dataset of Japanese signatures compared to the dataset of Dutch signatures, even though it was trained using Latin script signatures only. One possible explanation is that the task of verifying Japanese signatures—at least on the given data—is easier. This explanation finds support in the fact that the overall results in the SigWiComp2013 challenge are better on this dataset as well.

³ The “best ranked” systems from the challenge were selected based on their C_{llr}^{min} value. Other participants partly ranked higher in other values. Please refer to the original results in [44].

| <i>Metric</i> | <i>Dutch</i> | <i>Japanese</i> |
|----------------------------|----------------|-----------------|
| Accuracy at EER | 81.76 % | 93.39 % |
| FRR | 18.24 % | 6.66 % |
| FAR | 18.24 % | 6.57 % |
| Best Accuracy (threshold) | 81.99 % (66.6) | 93.57 % (36.5) |
| Best F-measure (threshold) | 82.78 % (72.4) | 93.64 % (36.5) |

Table 7.2: Application dependent evaluation results for the SigWiComp2013 “Dutch Offline Signatures” dataset and the SigWiComp2013 “Japanese Offline Signatures” dataset.

| <i>Metric</i> | <i>Dutch</i> | <i>Japanese</i> |
|--------------------|--------------|-----------------|
| C_{llr} | 0.705 923 5 | 0.421 013 6 |
| C_{llr}^{min} | 0.653 740 5 | 0.316 641 5 |
| $\Delta_{C_{llr}}$ | 0.052 183 0 | 0.104 372 1 |

Table 7.3: C_{llr} , C_{llr}^{min} , and their difference ($\Delta_{C_{llr}}$) for the SigWiComp2013 “Dutch Offline Signatures” dataset and the SigWiComp2013 “Japanese Offline Signatures” dataset.

| <i>ID in [44] or our system</i> | <i>Accuracy</i> | <i>FRR</i> | <i>FAR</i> | C_{llr} | C_{llr}^{min} |
|-------------------------------------|-----------------|----------------|----------------|------------------|------------------|
| 2 | 76.83 % | 23.70 % | 23.10 % | 0.880 048 | 0.642 632 |
| <i>Signature Embedding</i> | 81.76 % | 18.24 % | 18.24 % | 0.705 924 | 0.653 741 |
| 4 | 74.93 % | 25.19 % | 25.05 % | 0.979 237 | 0.698 044 |
| 3 | 75.56 % | 24.44 % | 24.44 % | 1.086 197 | 0.706 733 |
| 10 | 72.95 % | 27.41 % | 27.00 % | 1.023 746 | 0.728 198 |
| 6 | 72.14 % | 27.41 % | 27.93 % | 0.966 780 | 0.739 129 |

Table 7.4: Comparison of Signature Embedding to ICDAR SigWiComp2013 Task 1: Dutch Offline Signature Verification

| <i>ID in [44] or our system</i> | <i>Accuracy</i> | <i>FRR</i> | <i>FAR</i> | C_{llr} | C_{llr}^{min} |
|-------------------------------------|-----------------|---------------|---------------|------------------|------------------|
| <i>Signature Embedding</i> | 93.39 % | 6.66 % | 6.57 % | 0.421 014 | 0.316 642 |
| 2 | 90.72 % | 9.74 % | 9.72 % | 0.796 040 | 0.339 265 |
| 3 | 89.82 % | 10.23 % | 10.14 % | 0.814 598 | 0.349 146 |
| 4 | 86.95 % | 13.04 % | 13.06 % | 0.831 630 | 0.400 977 |
| 6 | 76.70 % | 23.60 % | 23.06 % | 0.980 174 | 0.665 241 |
| 10 | 74.59 % | 25.41 % | 25.42 % | 1.002 516 | 0.694 036 |

Table 7.5: Comparison of Signature Embedding to ICDAR SigWiComp2013 Task 2: Japanese Offline Signature Verification

7.5 Experiments

In this section, we discuss a number of experiments, evaluating the impact of a selection of design decisions on the systems' performance. Specifically, we discuss a further data augmentation method, the implications of the *hard triplets* hyperparameter, and the effect number of reference signatures available for making a decision.

7.5.1 Augmentation with a Background

As an additional method of data augmentation, we added a paper structure-like background to the training samples.

We consider this augmentation realistic, since in a real world application the system would be required to operate on handwritten signatures captured from scanned or photographed paper. The signature samples do not show the signature on a clean white background, but include artifacts like reference lines and the structure of the paper. It is typical for deep learning approaches that such artifacts are not removed before the data is input into the DNN. Instead, the DNN should be trained to abstract from irrelevant information.

In order to train the DNN with samples containing non-white background, we extended our data augmentation scripts to create samples with different backgrounds. The samples are created as follows: At random, an image of paper or paper-like structure is selected from a predefined set of images. From the selected image, a randomly selected patch, fitting the signature in size and scale, is cropped. The signature is then placed in the center of the patch of background. Figure 7.7 shows a selection of samples generated in this manner.

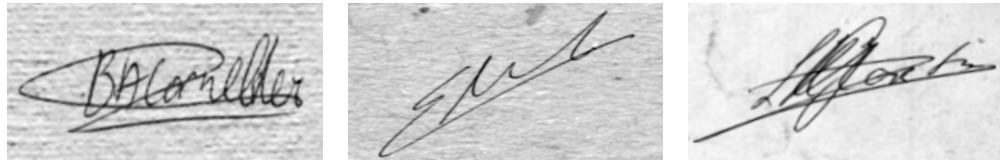


Figure 7.7: Three signature samples from the SigWiComp2013 Dutch dataset augmented by different background structures.

We trained our system using the samples augmented by this process. Our experiments revealed that the model can be trained just as well as it could without the additional background. This indicates that the DNN is well capable of extracting the information relevant to signature verification from these samples, while ignoring the irrelevant background. Nonetheless, in the training for our evaluation and comparison in [chapter 7](#) we did not make use of this augmentation method, since the samples in the evaluation sets exclusively contain clean white backgrounds.

7.5.2 Ratio of Hard Triplets

The ratio of hard triplets is a parameter specific to our approach and system, which determines how many triplets in a batch use skilled forgeries as negative sample.

The hard triplets are the triplets on which the DNN model learns to fulfill its actual task: detecting skilled forgeries. The other triplets in the batch contain random forgeries as negative samples. From these easy triplets the model learns to detect very poor (random) forgeries and to distinguish between users. As a result, the model could be employed for identification of users as well.

The task addressed in this work, however, is the verification of signatures. Hence, we typically chose a high ratio of hard triplets. In our final model, which was evaluated above, the ratio of hard triplets was set to 90 %.

The fact that identification is not a requirement to the system motivates the experiment to not use any easy triplets at all. Figure 7.8 shows plots of the in-training validation metrics for this setup. Note that all other conditions for the training of this model were exactly the same as for the model discussed in chapter 7. That is, we used the same datasets, based the training on the same pretrained model, employed the same neural network design, and left the other hyperparameters untouched.

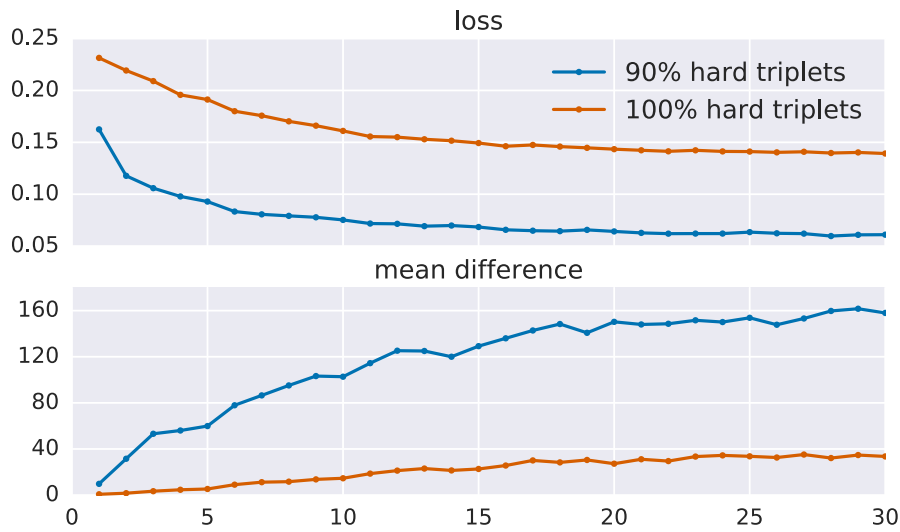


Figure 7.8: Validation loss and validation mean difference for models trained with 90 %, and with 100 % hard triplets.

From figure 7.8 it becomes clear that the model trained using hard triplets only performs worse than our final model. In other words, the 10 % share of easy triplets we used in our original model has advantageous effects.

As table 7.6 shows, this is also revealed by the accuracy and the optimized C_{llr} . A possible interpretation is, that the simpler task of distinguishing different users' signatures already teaches the model to extract some of the relevant features.

| <i>Hard Triplet Ratio</i> | <i>Accuracy at EER</i> | C_{llr}^{min} |
|---------------------------|------------------------|-----------------|
| 90 % | 81.50 % | 0.658 720 |
| 100 % | 79.08 % | 0.687 038 |

Table 7.6: The system trained with 90 % hard triplets reaches both better accuracy and better optimized C_{llr} . Both systems had been trained for 30 epochs when the above metrics were obtained. 12 reference signatures were used.

Conceptually, this is similar to a pretraining on a related task, where the model’s parameters are initialized in a way that the convolutional layers already provide a somewhat meaningful feature extraction.

This hypothesis is supported by the PCA shown in [figure 7.9](#). The figure depicts signatures from the SigWiComp2013 Dutch dataset, embedded by the model that was trained with hard triplets only. It becomes apparent that different users’ genuine signatures form clusters, even though the model was never explicitly trained to distinguish them. Consequently, the features the model extracts in order to compare genuine signature to forgeries are also suitable to compare one user’s signatures to those of another user. It is hence likely that features learned on random forgeries are also useful in order to distinguish genuine signatures from skilled forgeries.

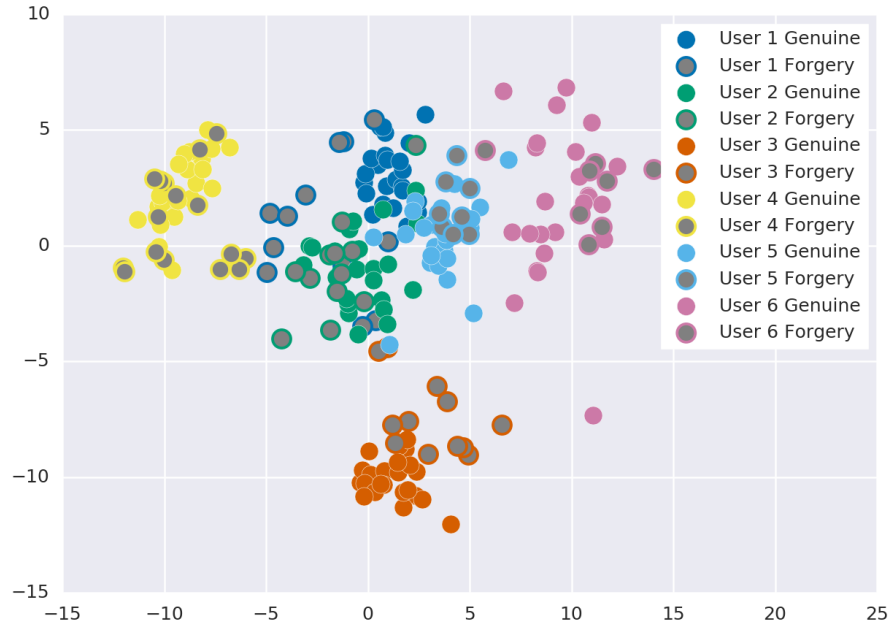


Figure 7.9: Trained with hard triplets only, the DNN still embeds genuine signatures clustered by user. This figure shows a PCA of embedded Dutch signatures.

7.5.3 Number of Reference Signatures

The number of available reference signatures has a large impact on the system's performance. If more than one reference signature are available, we compute the centroid of their embeddings. Thus, we obtain a more accurate reference where genuine signatures of the user in question should be embedded, the more reference signatures are available. The system can hence provide more accurate predictions about the genuineness of a questioned signature.

Consider the case where only one reference signature is available. If this signature is an outlier then it will be embedded comparably far from other genuine signatures of the same user. The distance of questioned signatures that are genuine will hence be larger. As a result, the rate of false rejects is expected to increase.

| # Reference Signatures | Threshold at EER | Accuracy | FRR | FAR | C_{llr} | C_{llr}^{min} |
|------------------------|------------------|----------|---------|---------|-----------|-----------------|
| 1 | 102.6 | 71.80 % | 28.23 % | 28.19 % | 0.870 160 | 0.851 834 |
| 2 | 74.5 | 72.57 % | 27.41 % | 27.45 % | 0.862 929 | 0.843 499 |
| 6 | 64.4 | 79.87 % | 20.16 % | 20.10 % | 0.744 889 | 0.702 132 |
| 12 | 61.6 | 81.76 % | 18.24 % | 18.24 % | 0.705 923 | 0.653 740 |
| 18 | 60.9 | 82.62 % | 17.37 % | 17.38 % | 0.692 097 | 0.635 480 |

Table 7.7: Application dependent and independent evaluation metrics for different numbers of reference signatures. The model's performance improves in all metrics, if more reference signatures are available.

Table 7.7 reports both application dependent and independent evaluation metrics our model produced on the "SigWiComp2013 Dutch" dataset for different numbers of reference signatures. The table states the threshold at which the EER was produced and the accuracy, FRR, and FAR obtained for this threshold. Furthermore, C_{llr} and C_{llr}^{min} are indicated.

As expected, all the model produces better results on all metrics the more reference signatures are available. Moreover, it is insightful to observe the threshold at which the EER is produced: it decreases with an increasing amount of reference signatures. Intuitively, this makes sense. The clusters of one user's genuine signatures generated by the centroids of reference signatures are more dense than the clusters of the individual reference signatures. The system can thus employ a smaller threshold to distinguish genuines from forgeries.

8 Related Work

In this chapter we discuss related work to this thesis. We mostly focus on off-line signature verification, but broaden the scope to on-line signature verification where it is relevant.

This chapter is split to two sections. The first section is concerned with systems that rely on the extraction of handcrafted features that are used to train a classifier. The second section is concerned with deep learning based systems that rely on automatic feature extraction, similar to the system we propose.

8.1 Handcrafted Features

In this section we discuss systems that make use of handcrafted features. Such systems usually involve three phases of processing: **preprocessing**, **feature extraction**, and **classification**—either in order to train or in order to apply the system. The design of these features and the processing of the image that is required to extract them usually represent the core contribution of these works. In a deep learning based system, such as the one presented in this thesis, the design of features is omitted and preprocessing is only necessary in a limited way. A direct comparison is hence not always possible. In the following we will walk through each of the three phases and present how they have been implemented in literature.

Not all of the systems discussed here are writer independent. We indicate writer dependence or independence in the description of the classification phase.

Preprocessing

The very first step is to extract the signature from a given image. This involves locating the signature and removing the background, for example for signatures on banking checks. Dimauro et al. [16] located the signature by its well-known position on the check and removed the background by binarization, using a variant of Otsu’s method [55]. This step is, however, often not considered part of the signature verification system: in most conference challenges the datasets consist of already extracted and cleaned signature images [24].

Signature extraction is usually followed by the removal of noise that can result from the process of digitalizing the signature. Huang and Yan [29] use median-filtering and average filtering, as well as morphological operations—erosion and dilation—in order to remove noise.

Further preprocessing steps include scaling, centering, and cropping the signature. For centering, the gray-level centroid of the signature can be used [29]. Yilmaz

et al. [83] also transferred the signature image into a polar coordinate system with the signature's centroid as the center.

Of these steps cropping and resizing are the only steps employed in our system, as the DNN operates on a fixed input image size. Cropping can be done in a way that the signature exactly fits the image without any borders [23], or—as we do—leaving a white margin around the signature [29].

Feature Extraction

As for the phase of feature extraction, a variety of approaches has been explored. The approaches can be categorized as local or global, and pseudo-dynamic or static [59]. Local feature extraction approaches extract features from patches of the image, rather than from the whole image, which characterizes the global approach. Pseudo-dynamic feature extraction describes approaches that attempt to reconstruct information from the signing process as it would be available in an on-line signature verification scenario, for example inferring the pen velocity from the width of the stroke. Static approaches, in contrast, do not make use of such features.

A widely employed group of feature extraction approaches is categorized as *geometric*. Huang and Yan [29] made use of features such as the signature's skeleton and outline. In addition, they extracted high pressure regions as a pseudo-dynamic feature. Other commonly used geometric features include the width, height, and aspect-ratio of the signature. Baltzakis and Papamarkos [1] counted the number of endpoints and the number of closed loops in the signature as a feature.

Geometric features are often localized by segmenting them with an overlaid grid [1, 29, 35]. Huang and Yan [30] subsequently extracted the pixel density from each cell in the grid.

As an alternative to overlaying a grid, some authors examine the signature at multiple scales. Rivard et al. [60] analyzed the directional probability density function (PDF) at multiple scales, Zhang [85] proposed the use of a *pyramid histogram of oriented gradients* (HOG)—“a histogram of edge orientations computed for each image sub-region” [85]—at multiple scales.

Yilmaz et al. [83] combined HOG and local binary patterns (LBP), which computes histograms of a pixels neighborhood. With this approach and SVM classification, they achieved the highest score in the ICDAR SigWiComp 2013 challenge [44].

Many further groups of feature extraction approaches have been explored: *Graphometric* features [5, 35, 53] employ graphology—the analysis of handwriting in order to infer information about the character and psyche of its author. *Interest point matching* employs feature detection and description algorithms such as *SIFT* [62] and *SURF* [45].

Classification

The approach most closely related to the one presented in this thesis is the work of Bromley et al. [6]. The system they proposed is a writer dependent on-line

signature verification system. However, their “Siamese” classifier architecture is the first application of similarity metrics to the problem of signature verification that we are aware of. This classifier is discussed in greater detail in [section 3.4](#).

Eskander et al. [19] proposed a hybrid writer independent and writer dependent system. The main idea is to employ a writer independent classifier for new users. Once a user has authenticated against the system often enough and a sufficient number of signature samples has been collected, the system switches to a writer dependent classifier. For the writer independent classifier, a dissimilarity representation is derived from the handcrafted features. The requirement to the resulting similarity metric is again that genuine signatures of one user are embedded close to each other in the target space, while forgeries for that user are embedded far away from the genuines. In order to foster representations that comply to this requirement, Eskander et al. performed a boosting feature selection (BFS) [73]. In contrast, in our system this requirement can be modeled as part of the loss computation.

A different approach to classification is the *ensemble of classifiers*. Bertolini et al. [5] proposed a writer independent system based on this approach. They introduced a new set of graphometric features. They extracted these features locally at various scales in order to create a large number of classifiers. Subsequently, they employed genetic algorithms in order to select the best performing classifiers, optimizing for (a) the area under the ROC curve and (b) the true positive rate at a given false positive rate.

8.2 Deep Learning

Deep learning techniques have not been widely used in the field of signature verification [24]. This fact was part of the motivation for this thesis.

Similar to the method proposed in this thesis, Khalajzadeh et al. [36] used a DNN for feature extraction. However, the authors do not make use of dissimilarity metrics. Instead, they train one model for each of the 22 users in their experiment, performing a writer dependent classification. They achieved a very high accuracy, but considered random forgeries only.

In their master thesis, Drott and Hassan-Reza [17] trained a DNN for writer dependent on-line signature verification. Given this scenario, they reached a very high true positive rate (96.7%) and a very low false positive rate (0.6%)

Most recently, Hafemann et al. [25] proposed the use of a DNN for writer independent off-line signature verification. Their setup is similar to the setup in this thesis: In order to ensure writer independence, the feature extraction using the DNN is learned on a training set that does not include any authors from the evaluation set. The trained DNN is used to obtain a feature representation of each signature in the evaluation set. Based on these feature representations, a binary (“genuine” or “forged”) classifier is trained.

The difference to the approach we propose is that the feature representation the authors obtain cannot be used as a similarity metric. The training of their DNN is

based on cross-entropy loss. Thus, no requirement concerning how the embedding should be done can be modeled, which is why the additional training of the binary classifier is required. Furthermore, the binary classifier provides a hard decision, rather than a soft decision, which would be more desirable in some scenarios (we discussed hard and soft decisions in [section 5.4](#)).

In addition to the writer independent component, the system of Hafemann et al. is equipped with a writer dependent component, which is able to leverage the feature representations obtained by the DNN. Their proposed system is thus a hybrid system similar to the one presented by Eskander et al. [19].

From a technological point of view, Schroff et al. [65] applied an approach very similar to ours to the problem of face recognition. They used a DNN in order to learn a similarity metric of faces. Their training is also based on embedding triplets of *anchors*, *positives* and *negatives*.

9 Conclusion

In this thesis we investigated how deep learning techniques, specifically deep neural networks (DNNs) and deep metric learning, can be applied to the domain of signature verification.

We presented a new approach to the task of writer independent off-line signature verification that is based on a deep learned similarity metric. This approach allows us to compare two given signatures based on an embedding in a high-dimensional space, in order to confirm or to refute that both signatures are created by the same author.

We implemented a system that makes use of the *Signature Embedding* approach. The system we created is built around a DNN that performs the embedding of signatures. In order to train the DNN, the system employs a triplet-based loss function, that compares the embeddings of two genuine signatures of a user to the embedding of a forged signature. This way, the distance between the genuine signatures is minimized, while the distance of a genuine to the forged signature is maximized.

In the resulting high-dimensional embedding, the Euclidean distance between two signatures functions as a metric of their similarity, and thus as an assessment of whether or not the signatures have been created by the same person. Given this embedding, we investigated how both hard and soft decisions can be derived.

We explored which datasets can be used for the creation of such a system and how they need to be prepared and augmented. We further investigated how the training progress of such a system can be validated during training and provided an overview of metrics that can be employed in order to evaluate the performance of the final system. Based on these metrics, we evaluated our system's performance on the ICDAR SigWiComp 2013 [44] challenge on off-line signature verification. We found that it outperforms the systems that participated in the challenge in almost all respects.

In this thesis we also provided an introduction into the domain biometric authentication and the task of signature verification, and discussed these technologies from an ethical point of view. Furthermore we provided an overview to the theoretical background in machine learning and similarity metrics that is required for this work. We reviewed our approach in the context of related work on the problem of signature verification and compared it to the approaches in literature.

In the remainder of this chapter we want to highlight a number of learnings and insights we gained during our work on this project, and provide a perspective on application scenarios and possible improvements of the approach.

9.1 Learnings and Insights

Our work on this project gave us the opportunity to explore deep learning technologies in great detail. With these technologies, the need for careful, manual engineering of features is largely omitted.

On the other hand, the need to carefully engineer the design of the DNN and its hyperparameters arises. Both of these are crucial elements in the overall system conception and have a major impact onto the system's performance. Moreover, many decisions taken in the design of such a system have to be evaluated empirically. In reward, the system gains high generality—we were able to employ the system to signatures of a script that it had never been trained on.

During the design and implementation of our system we found the *MNIST* dataset of handwritten digits very useful for debugging. The task of recognizing handwritten digits is not too closely related to the task of signature verification. Nonetheless, attempting to solve this comparably simple task on a well-known and ample dataset allowed us to discern which faults arose from our implementation and which from the employed datasets. As a result, both kinds of faults could be located and corrected more easily.

We furthermore want to emphasize the importance of good metrics for measurable progress. We found evaluation frameworks discussed and employed in literature most vital. In contrast, we found that visualizations, such as the PCA, are very useful in order to provide an intuition, but are too inaccurate to be employed as a validation metric.

9.2 Future Work

There are various options to investigate possible technical improvements to our system. The design of the DNN is one such option. We tried a variety of network designs, inspired from different works in literature, before we settled for the VGG-derived design that we ultimately employed. However, DNNs are a fast moving research area at this time and many new ideas on how to design a DNN are emerging. Specifically, we would like to employ the latest incarnation [71] of the *Inception* architecture [70], which shows very promising results especially in the domain of image recognition.

We believe that our system can be employed to tasks other than off-line signature verification as well, without any major modifications. A specifically suitable task seems to be handwriting recognition, as it is somewhat related to signature verification. Handwriting recognition has been a task in ICDAR SigWiComp challenges in recent years. In order to employ *Signature Embedding* on that task, only the assembly of triplets would have to be adjusted to the new task. Rather than comparing two genuine signatures with a forgery, the system would be trained by comparing two handwriting samples of one user to a handwriting sample of another user. We are motivated to see how our system performs in this domain.

Bibliography

- [1] H Baltzakis and N Papamarkos. “A new signature verification technique based on a two-stage neural network classifier”. In: *Engineering applications of Artificial intelligence* 14.1 (2001), pp. 95–103.
- [2] Luana Batista, Eric Granger, and Robert Sabourin. “Dynamic selection of generative–discriminative ensembles for off-line signature verification”. In: *Pattern Recognition* 45.4 (2012), pp. 1326–1340.
- [3] Ian Goodfellow Yoshua Bengio and Aaron Courville. “Deep Learning”. Book in preparation for MIT Press. 2016.
- [4] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. “Advances in optimizing recurrent networks”. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 8624–8628.
- [5] Diego Bertolini, Luiz S Oliveira, Edson Justino, and Robert Sabourin. “Ensemble of classifiers for off-line signature verification”. In: *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*. IEEE. 2008, pp. 283–288.
- [6] Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. “Signature verification using a “Siamese” time delay neural network”. In: *International Journal of Pattern Recognition and Artificial Intelligence* 7.04 (1993), pp. 669–688.
- [7] Niko Brümmer and Johan du Preez. “Application-independent evaluation of speaker detection”. In: *Computer Speech & Language* 20.2 (2006), pp. 230–275.
- [8] Niko Brümmer and David A Van Leeuwen. “On calibration of language recognition scores”. In: *2006 IEEE Odyssey-The Speaker and Language Recognition Workshop*. IEEE. 2006, pp. 1–8.
- [9] Augustin Cauchy. “Méthode générale pour la résolution des systemes d’équations simultanées”. In: *Comp. Rend. Sci. Paris* 25.1847 (1847), pp. 536–538.
- [10] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. “cudnn: Efficient primitives for deep learning”. In: *arXiv preprint arXiv:1410.0759* (2014).
- [11] Sumit Chopra, Raia Hadsell, and Yann LeCun. “Learning a similarity metric discriminatively, with application to face verification”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 1. IEEE. 2005, pp. 539–546.

- [12] Computer Committee for Information, communication Policy - Working Party on Information Security, and Privacy. *Biometric-based technologies*. Tech. rep. Available from: <http://www.oecd.org/sti/security-privacy>; last accessed 03.08.2016. Organisation for Economic Co-operation and Development (OECD), 2004.
- [13] Federal Financial Institutions Examination Council. "Authentication in an internet banking environment". In: *Financial Institution Letter, FIL-103-2005*. Washington, DC: Federal Deposit Insurance Corp.(FDIC). Retrieved March 18 (2005), p. 2005.
- [14] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. "Large scale distributed deep networks". In: *Advances in neural information processing systems*. 2012, pp. 1223–1231.
- [15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255.
- [16] Giovanni Dimauro, Sebastiano Impedovo, Giuseppe Pirlo, and A Salzo. "Automatic bankcheck processing: A new engineered system". In: *International Journal of Pattern Recognition and Artificial Intelligence* 11.04 (1997), pp. 467–504.
- [17] Beatrice Drott and Thomas Hassan-Reza. "On-line Handwritten Signature Verification using Machine Learning Techniques with a Deep Learning Approach". MA thesis. Lund University, 2015.
- [18] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.
- [19] George S Eskander, Robert Sabourin, and Eric Granger. "Hybrid writer-independent–writer-dependent offline signature verification system". In: *IET biometrics* 2.4 (2013), pp. 169–181.
- [20] Data Protection Working Party of the European commission. *Biometrics*. Tech. rep. Available from: http://ec.europa.eu/justice/policies/privacy/docs/wpdocs/2003/wp80_en.pdf; accessed 03.08.2016. Brussels: EC, 2003.
- [21] Miguel A Ferrer, Moises Diaz-Cabrera, and Aythami Morales. "Static signature synthesis: A neuromotor inspired approach for biometrics". In: *IEEE transactions on pattern analysis and machine intelligence* 37.3 (2015), pp. 667–680.
- [22] Stuart Geman, Elie Bienenstock, and René Doursat. "Neural networks and the bias/variance dilemma". In: *Neural computation* 4.1 (1992), pp. 1–58.

- [23] Samaneh Ghandali and Mohsen Ebrahimi Moghaddam. "A method for off-line Persian signature identification and verification using DWT and image fusion". In: *2008 IEEE International Symposium on Signal Processing and Information Technology*. IEEE. 2008, pp. 315–319.
- [24] Luiz G Hafemann, Robert Sabourin, and Luiz S Oliveira. "Offline handwritten signature verification-literature review". In: *arXiv preprint arXiv:1507.07909* (2015).
- [25] Luiz G Hafemann, Robert Sabourin, and Luiz S Oliveira. "Writer-independent Feature Learning for Offline Signature Verification using Deep Convolutional Neural Networks". In: *arXiv preprint arXiv:1604.00974* (2016).
- [26] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554.
- [27] Elad Hoffer and Nir Ailon. "Deep metric learning using triplet network". In: *International Workshop on Similarity-Based Pattern Recognition*. Springer. 2015, pp. 84–92.
- [28] Harold Hotelling. "Analysis of a complex of statistical variables into principal components." In: *Journal of educational psychology* 24.6 (1933), p. 417.
- [29] Kai Huang and Hong Yan. "Off-line signature verification based on geometric feature extraction and neural network classification". In: *Pattern Recognition* 30.1 (1997), pp. 9–17.
- [30] Kai Huang and Hong Yan. "Off-line signature verification using structural feature correspondence". In: *Pattern Recognition* 35.11 (2002), pp. 2467–2477.
- [31] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95.
- [32] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Synthetic data and artificial neural networks for natural scene text recognition". In: *arXiv preprint arXiv:1406.2227* (2014).
- [33] Anil K Jain, Arun Ross, and Salil Prabhakar. "An introduction to biometric recognition". In: *IEEE Transactions on circuits and systems for video technology* 14.1 (2004), pp. 4–20.
- [34] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed 2016-07-18]. 2001–present.
- [35] Edson JR Justino, Abdenain El Yacoubi, Flávio Bortolozzi, and Robert Sabourin. "An off-line signature verification system using HMM and graphometric features". In: *Fourth IAPR International Workshop on Document Analysis Systems (DAS), Rio de*. Citeseer. 2000, pp. 211–222.
- [36] Hurieh Khalajzadeh, Mohammad Mansouri, and Mohammad Teshnehlab. "Persian Signature Verification using Convolutional Neural Networks". In: *International Journal of Engineering Research and Technology*. Vol. 1. 2 (April-2012). ESRSA Publications. 2012.

- [37] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).
- [38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems* 25. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105.
- [39] Yann LeCun, Corinna Cortes, and Christopher JC Burges. "MNIST handwritten digit database". In: *AT&T Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist> (2010). On-line source and description of the dataset.
- [40] Marcus Liwicki, Muhammad Imran Malik, C Elisa van den Heuvel, Xiaohong Chen, Charles Berger, Reinoud Stoel, Michael Blumenstein, and Bryan Found. "Signature verification competition for online and offline skilled forgeries (SigComp2011)". In: *2011 International Conference on Document Analysis and Recognition*. IEEE. 2011, pp. 1480–1484.
- [41] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of Machine Learning Research* 9.Nov (2008), pp. 2579–2605.
- [42] James MacQueen et al. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297.
- [43] Muhammad Imran Malik, Sheraz Ahmed, Angelo Marcelli, Umapada Pal, Michael Blumenstein, Linda Alewijns, and Marcus Liwicki. "ICDAR2015 competition on signature verification and writer identification for on-and off-line skilled forgeries (SigWlcomp2015)". In: *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*. IEEE. 2015, pp. 1186–1190.
- [44] Muhammad Imran Malik, Marcus Liwicki, Linda Alewijnse, Wataru Ohyama, Michael Blumenstein, and Bryan Found. "Icdar 2013 competitions on signature verification and writer identification for on-and offline skilled forgeries (sigwicomp 2013)". In: *2013 12th International Conference on Document Analysis and Recognition*. IEEE. 2013, pp. 1477–1483.
- [45] Muhammad Imran Malik, Marcus Liwicki, Andreas Dengel, Seiichi Uchida, and Volkmar Frinken. "Automatic signature stability analysis and verification using local features". In: *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*. IEEE. 2014, pp. 621–626.
- [46] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.
- [47] Marvin Minsky and Seymour Papert. "Perceptrons." In: (1969).

- [48] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.
- [49] Emilio Mordini and Carlo Petrini. "Ethical and social implications of biometric identification technology." In: *Annali dell'Istituto superiore di sanita* 43.1 (2006), pp. 5–11.
- [50] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 807–814.
- [51] Vu Nguyen, Michael Blumenstein, and Graham Leedham. "Global features for the off-line signature verification problem". In: *2009 10th International Conference on Document Analysis and Recognition*. IEEE. 2009, pp. 1300–1304.
- [52] Michael A Nielsen. *Neural networks and deep learning*. On-line book <http://neuralnetworksanddeeplearning.com/>. Determination Press, 2015.
- [53] Luiz S Oliveira, Edson Justino, Cinthia Freitas, and Robert Sabourin. "The graphology applied to signature verification". In: *12th Conference of the International Graphonomics Society*. 2005, pp. 286–290.
- [54] Javier Ortega-Garcia, J Fierrez-Aguilar, D Simon, J Gonzalez, M Faundez-Zanuy, V Espinosa, A Satue, I Hernaez, J-J Igarza, C Vivaracho, et al. "MCYT baseline corpus: a bimodal biometric database". In: *IEE Proceedings-Vision, Image and Signal Processing* 150.6 (2003), pp. 395–401.
- [55] Nobuyuki Otsu. "A threshold selection method from gray-level histograms". In: *Automatica* 11.285-296 (1975), pp. 23–27.
- [56] John D Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E Lefohn, and Timothy J Purcell. "A survey of general-purpose computation on graphics hardware". In: *Computer graphics forum*. Vol. 26. 1. Wiley Online Library. 2007, pp. 80–113.
- [57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [58] Réjean Plamondon and Sargur N Srihari. "Online and off-line handwriting recognition: a comprehensive survey". In: *IEEE Transactions on pattern analysis and machine intelligence* 22.1 (2000), pp. 63–84.
- [59] Dominique Rivard. "Multi-feature approach for writer-independent offline signature verification". MA thesis. École de technologie supérieure, 2010.
- [60] Dominique Rivard, Eric Granger, and Robert Sabourin. "Multi-feature extraction and selection in writer-independent off-line signature verification". In: *International Journal on Document Analysis and Recognition (IJ DAR)* 16.1 (2013), pp. 83–103.

- [61] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.
- [62] Javier Ruiz-del-Solar, Christ Devia, Patricio Loncomilla, and Felipe Concha. "Offline signature verification using local interest points and descriptors". In: *Iberoamerican Congress on Pattern Recognition*. Springer. 2008, pp. 22–29.
- [63] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3 (1988), p. 1.
- [64] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*. Vol. 2. Prentice hall Upper Saddle River, 2003.
- [65] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015.
- [66] Matthew Schultz and Thorsten Joachims. "Learning a distance metric from relative comparisons". In: *Advances in neural information processing systems (NIPS)* (2004), p. 41.
- [67] Patrice Y Simard, David Steinkraus, and John C Platt. "Best practices for convolutional neural networks applied to visual document analysis." In: *ICDAR*. Vol. 3. 2003, pp. 958–962.
- [68] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).
- [69] Irwin Sobel and Gary Feldman. "A 3x3 isotropic gradient operator for image processing". In: *a talk at the Stanford Artificial Project in* (1968), pp. 271–272.
- [70] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9.
- [71] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. "Rethinking the inception architecture for computer vision". In: *arXiv preprint arXiv:1512.00567* (2015).
- [72] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural Networks for Machine Learning* 4.2 (2012).
- [73] Kinh Tieu and Paul Viola. "Boosting image retrieval". In: *International Journal of Computer Vision* 56.1-2 (2004), pp. 17–36.

- [74] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. "Chainer: a next-generation open source framework for deep learning". In: *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*. 2015.
- [75] Matthew Turk and Alex Pentland. "Eigenfaces for recognition". In: *Journal of cognitive neuroscience* 3.1 (1991), pp. 71–86.
- [76] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation". In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.
- [77] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. "scikit-image: image processing in Python". In: *PeerJ* 2 (2014), e453.
- [78] David A Van Leeuwen and Niko Brümmer. "An introduction to application-independent evaluation of speaker recognition systems". In: *Speaker classification I*. Springer, 2007, pp. 330–353.
- [79] J Francisco Vargas, Miguel A Ferrer, Carlos M Travieso, and Jesús B Alonso. "Off-line handwritten signature GPDS-960 corpus". In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. 2007.
- [80] Cheng Wang, Haojin Yang, Christian Bartz, and Christoph Meinel. "Image Captioning with Deep Bidirectional LSTMs". In: *arXiv preprint arXiv:1604.00790* (2016).
- [81] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. "Learning fine-grained image similarity with deep ranking". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1386–1393.
- [82] Michael Waskom, Olga Botvinnik, drewokane, Paul Hobson, Yaroslav Halchenko, Saulius Lukauskas, Jordi Warmenhoven, John B. Cole, Stephan Hoyer, Jake Vanderplas, and et al. *seaborn: vo.7.0 (January 2016)*. 2016. DOI: [10.5281/zenodo.45133](https://doi.org/10.5281/zenodo.45133).
- [83] Mustafa Berkay Yilmaz, Berrin Yanikoglu, Caglar Tirkaz, and Alisher Kholmatov. "Offline signature verification using classifier combination of HOG and LBP features". In: *Biometrics (IJCB), 2011 International Joint Conference on*. IEEE. 2011, pp. 1–7.
- [84] Matthew D Zeiler. "ADADELTA: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701* (2012).
- [85] Bailing Zhang. "Off-line signature verification and identification by pyramid histogram of oriented gradients". In: *International Journal of Intelligent Computing and Cybernetics* 3.4 (2010), pp. 611–630.
- [86] YT Zhou and R Chellappa. "Computation of optical flow using a neural network". In: *Neural Networks, 1988., IEEE International Conference on*. IEEE. 1988, pp. 71–78.

